Compact Object Mergers: Population Astrophysics and Statistics

COMPAS is a platform for the exploration of populations of compact binaries formed through isolated binary evolution (Stevenson et al. (2017); Vigna-Gómez et al. (2018); Barrett et al. (2018); Neijssel et al. (2019); Broekgaarden et al. (2019); Stevenson et al. (2019); Chattopadhyay et al. (2020)). The COMPAS population synthesis code is flexible, fast and modular, allowing rapid simulation of binary star evolution. The complete COMPAS suite includes the population synthesis code together with a collection of tools for sophisticated statistical treatment of the synthesised populations.

Visit http://compas.science for more information.

Contents

Revision History

Date	Version	Description	Author
2012-2019		Original COMPAS manual	Team COMPAS
1 September 2019	0.1	Initial draft	Jeff Riley
20 September 2019	0.2	Updated compilation requirements	Jeff Riley
1 October 2019	0.3	Added (minimal) CHE documentation	Jeff Riley
21 October 2019	0.4	Added Grids documentation	
		Added Programming Style and Conventions	
		Added Appendices A-E	
		Reformatted for User Guide, Developer Guide etc.	Jeff Riley
18 December 2019	0.5	Updated Grids documentation for kick values	Jeff Riley
20 December 2019	0.6	Updated Appendices A, B, D, E	Jeff Riley
23 January 2020	0.7	Updated Grids documentation for kick values	Jeff Riley
31 March 2020	1.0	Updated for Beta release	Jeff Riley
22 April 2020	1.1	Updated to reflect pre2ndSN $\rightarrow$ preSN variable changes	Simon Stevenson
27 April 2020	1.2	Updated to reflect SSE Grid file changes	Jeff Riley
7 September 2020	2.0	Created LaTex version	Jeff Riley
9 September 2020	2.1	Added SSE & BSE Switch Log files documentation	
		Added SSE Supernova log file documentation	Jeff Riley
11 September 2020	2.2	Added description for RLOF output parameters	Mike Lau
3 November 2020	2.3	Updated for new grid file functionality	Jeff Riley
4 November 2020	2.4	Updated for changed switchlog functionality	Jeff Riley
10 November 2020	2.5	Updated for added SSE System Parameters file	
		Clean up/fix for past code changes	Jeff Riley
10 November 2020	2.6	Added description of Schneider remnant mass and MassTransferDonorHistory	Reinhold Willcox
13 November 2020	2.7	Added description of metallicity-distribution and related options	
		Removed references to AIS; minor fixes and cleanup	Jeff Riley
19 November 2020	2.8	Added description of option luminous-blue-variable-prescription.	
		Added description of option mass-ratio.	
		A few typos and omissions resolved.	Jeff Riley
27 November 2020	2.9	Added description of option overall-wind-mass-loss-multiplier.	Lieke van Son
9 December 2020	2.10	Added description of option check-photon-tiring-limit and dominant mass loss variable.	Tom Wagg
14 December 2020	2.11	Changed architecture description to include newly added Remnants and WhiteDwarfs classes.	
		Changed pseudo-code descriptions of both SSE and BSE evolution algorithms to flow charts. Updated flow charts to better describe the algorithms.	Jeff Riley
18 December 2020	2.12	Removed references to Mass_Transfer_Case_Initial parameter	Reinhold Willcox
08 January 2021	2.13	Added (brief) description of HDF5 logfile support.	
		Added description of option hdf5-buffer-size.	
		Added description of option hdf5-chunk-size.	
		Added description of option logfile-type.	
		Removed description of, and references to, option logfile-delimiter.	Jeff Riley
19 February 2021	2.14	Added descriptions for options rotational-frequency, rotational-frequency-1, and rotational-frequency-2	Jeff Riley
06 April 2021	2.15	Added FARMER prescription for option pulsational-pair-instability-prescription	Lieke van Son
20 April 2021	2.16	Added option add-options-to-sysparms	Jeff Riley
18 May 2021	2.17	Changed default LBV prescription	Tom Wagg

User Guide

Installation instructions for COMPAS and its dependencies are shown in the COMPAS Getting Started guide provided in the COMPAS suite. Please refer to that guide for dependency requirements etc.

COMPAS Input

COMPAS provides wide-ranging functionality and affords users much flexibility in determining how the synthesis and evolution of stars (single or binary) is conducted. Users configure COMPAS functionality and provide initial conditions via the use of program options and grid files.

Program Options

COMPAS provides a rich set of configuration parameters via program options, allowing users to vary many parameters that affect the evolution of single and binary stars, and the composition of the population of stars being evolved. Furthermore, COMPAS allows some parameters to be specified as ranges, or sets, of values via the program options, allowing users to specify a grid of parameter values on the commandline. Combining commandline program options ranges and sets with a grid file allows users more flexibility and the ability to specify more complex combinations of parameter values.

Not all program options can be specified as ranges or sets of values. Options for which mixing different values in a single execution of COMPAS would either not be meaningful or might cause undesirable results, such as options that specify the mode of evolution (e.g. --*mode*), or the name or path of output files (e.g. --*output-path*, --*logfile-detailed-output* etc.), cannot be specified as a range or set of values. COMPAS will issue an error message if ranges or sets are specified for options for which they are not supported.

The full list of program options provided by COMPAS is described in Appendix **Program Options**.

Program Option Ranges

A range of values can be specified for any numeric options (i.e. integer (or integer variant), and floating point (or floating point variant) data types) that are not excluded from range specifications (see note above).

Option value ranges are specified by

> --*option-name range-specifier*

where *range-specifier* is

> *range-identifier*[*start*,*count*,*increment*]

and

range-identifier	is one of {'r', 'range'} (case is not significant)
start	is the starting value of the range
count	is the number of values in the range (must be an *unsigned long int*)
increment	is the amount by which the value increments for each value in the range

Note that

range-identifier is optional for *range-specifier*.
start and *increment* must be the same data type as *option-name*.

count must be a positive integer value.

To specify a range of values for the *--metallicity* option, a user, if running COMPAS from the commandline and with no grid file, would type any of the following:

./COMPAS --metallicity [0.0001,5,0.0013]
./COMPAS --metallicity r[0.0001,5,0.0013]
./COMPAS --metallicity range[0.0001,5,0.0013]

In each of the examples above the user has specified, by the use of the *range-specifier*, that five binary stars should be evolved, with constituent star metallicities = 0.0001, 0.0014, 0.0027, 0.0040, and 0.0053.

To evolve a grid of binaries with ten different metallicities, starting at 0.0001 and incrementing by 0.0002, and five different common envelope alpha values, starting at 0.1 and incrementing by 0.2, the user would type

./COMPAS --metallicity [0.0001,10,0.0013] --common-envelope-alpha [0.1,5,0.2]

and COMPAS would evolve a grid of 50 binaries using the 10 metallicity values and 5 common envelope alpha values.

Program Option Sets

A set of values can be specified for options of any data type that are not excluded from set specifications (see note above).

Option value sets are specified by

--option-name set-specifier

where *set-specifier* is

set-identifier[$value_1$,$value_2$,$value_3$, ... ,$value_n$]

and

set-identifier	is one of {'s', 'set'} (case is not significant)
$value_i$	is a value for the option

Note that

set-identifier is mandatory for *set-specifier*.
$value_i$ must be the same data type as *option-name*.
Valid values for boolean options are {1|0, TRUE|FALSE, YES|NO, ON|OFF}, and all set values must be of the same type (i.e. all 1|0, or all YES|NO etc.).
There is no limit to the number of values specified for a set, values can be repeated, and neither order nor case is significant.

To specify a set of values for the *--eccentricity-distribution* option, a user, if running COMPAS from the commandline and with no grid file, would type any of the following:

./COMPAS --eccentricity-distribution s[THERMALISED,FIXED,FLAT]
./COMPAS --eccentricity-distribution set[THERMALISED,FIXED,FLAT]

In each of the examples above the user has specified, by the use of the *set-specifier*, that three binary stars should be evolved, using the eccentricity distributions 'THERMALISED', 'FIXED', and 'FLAT'.

Grid Files

A grid file allows users to specify initial values for multiple systems for both Single Star Evolution (SSE) and Binary Star Evolution (BSE) for: each line of a grid file is used by COMPAS to set the initial values of an individual single star (SSE) or binary star (BSE), and each single star or binary star defined by a grid file line is evolved using those initial values.

Each line of a grid file is a set of option specifications, with the specifications being exactly as they would appear on the commandline if running COMPAS from the commandline.

For example, a grid file could contain the following two lines:

> --metallicity 0.001 --eccentricity 0.0 --remnant-mass-prescription fryer2012
> --remnant-mass-prescription mullermandel --metallicity 0.02 --semi-major-axis 45.678

in which case COMPAS would evolve two binaries, with the option values set per the grid file lines.

Grid files can have blank lines and comments. Comments begin with a hash/pound character ('#') - the hash/pound character and text beyond it are ignored by COMPAS.

Not all program options can be specified in a grid file. Options that should remain constant for a single execution of COMPAS, such as options that specify the mode of evolution (e.g. --*mode*), or the name or path of output files (e.g. --*output-path*, --*logfile-detailed-output* etc.) can only be specified on the commandline. COMPAS will issue an error message if an option that is not supported in a grid file is specified on a grid file line.

Program Option Defaults

Any program options that are not specified take default values:

- On a grid file line, program options that are not explicitly specified default to the value specified for that option on the commandline.
- On the commandline, program options that are not explicitly specified default to the COMPAS default value for the option (as specified in the COMPAS code - may be sampled from a distribution).

This means that a program option not explicitly specified on a grid file line will take the value for that option as it was specified on the commandline, or the COMPAS default value if the option was not explicitly specified on the commandline. That is, the value for any option not specified on a grid file line option falls back to the value specified on the commandline, which falls back to the COMPAS default if it was not specified on the commandline.

Mixing Ranges and Sets

Ranges and sets can be specified together, and there is no limit to the number of ranges or sets that can be specified on the commandline, or in the grid file.

Consider the following grid file, named 'gridfile.txt':

> --metallicity r[0.0001,5,0.0013] --common-envelope-alpha s[0.1,0.2,0.6,0.9]
> --fryer-supernova-engine s[rapid,delayed] --eccentricity r[0.0001,3,0.0003]

Running COMPAS with

./COMPAS --grid gridfile.txt

would result in 26 binaries being evolved:

- 20 for the first grid line (5 for the range of metallicities, times 4 for the set of CE alpha values), and
- 6 for the second grid line (2 for the set of Fryer SN engine values, and 3 for the range of eccentricities)

In the example above, running COMPAS with

./COMPAS --remnant-mass-prescription s[mullermandel,fryer2012,hurley2000,muller2016] --grid gridfile.txt

would result in 104 binaries being evolved: the grid file would be 'executed' for each of the four remnant mass prescriptions specified on the commandline.

Multiple ranges and/or sets can be specified on the commandline, and on each line of the grid file – so very large numbers of stars/binaries can be evolved with just a few range/set specifications.

Specifying initial random seed values

The --*random-seed* option allows users to specify the initial value to be used to seed the pseudo-random number generator. Once set, the random seed values increments from its initial value for each star, or binary star, evolved. How the random seed increments depends upon the context.

The --*random-seed* option can be specified on either, or both, the commandline and a grid file line. If the option is not specified on one or the other, it defaults per the description above (see Section **Program Option Defaults**).

In general, if the --*random-seed* option is specified, the pseudo-random number generator will be seeded using the specified value for the first star, or binary star, evolved, then for each subsequent star or binary star, the seed value will be incremented by one and the pseudo-random number generator re-seeded. Seeding the pseudo-random number generator with a known seed for each star, or binary star, evolved ensures that the evolution of specific stars, or binary stars, can be reproduced.

Consider a single execution of COMPAS effected with

./COMPAS --random-seed 15 --number-of-systems 100 --metallicity 0.015

This would evolve 100 binary stars, each with metallicity = 0.015, and other initial attributes set to their defaults. The first of the 100 binary stars will be evolved using the random seed 15, the second 16, the third 17, and so on - each binary star will evolve using a unique random seed.

In the example shown above (see Section **Mixing Ranges and Sets**), all 104 binary stars would evolve with unique random seed values, ranging from 0 (the default, since the option was not specified on either the commandline or in the grid file), to 103.

In both these examples, the random seed was incremented in the context of the commandline. In the first example, the random seed was explicitly specified on the commandline, and in the second example the random seed defaulted to the commandline default.

Consider now a single execution of COMPAS, using the grid file 'mygrid.txt':

./COMPAS --random-seed 12 --grid mygrid.txt

where the contents of the grid file 'mygrid.txt' are

--allow-rlof-at-birth true --metallicity 0.1
--semi-major-axis 23.4 --random-seed 107
--random-seed 63 --metallicity 0.12 --eccentricity s[0.1,0.2,0.3,0.4]
--initial-mass-1 12.3

This would evolve 7 binary stars with random seed values 12, 107, 63, 64, 65, 66, and 18.

The first binary star evolved is the first line of the grid file. This line does not specify the *--random-seed* option, so the random seed defaults to the commandline value. The commandline did specify a value for the random seed (12), so that value is used. Since the first line of the grid file is the first binary star evolved, the random seed is not incremented, and the value of 12 is used.

The second binary star evolved is the second line of the grid file. This line does specify the *--random-seed* option. Since this is the first binary star evolved in the context of the random seed specified on the grid file line, the random seed is not incremented, and the value of 107 is used.

The third binary star evolved is the third line of the grid file. This line does specify the *--random-seed* option. Since this is the first binary star evolved in the context of the random seed specified on the grid file line, the random seed is not incremented, and the value of 63 is used.

The fourth, fifth, and sixth binary stars evolved are also from the third line of the grid file - a set of four values for *eccentricity* was specified. Since these are subsequent to the first binary star evolved in the context of the random seed specified on the grid file line, the random seed is incremented, and the values of 64, 65, and 66 are used.

The seventh binary star evolved is the fourth line of the grid file. This line does not specify the *--random-seed* option, so the random seed defaults to the commandline value. The commandline did specify a value for the random seed (12), so that value is used, but since this binary star is subsequent to the first binary star evolved in the context of the random seed specified on the commandline, the random seed is incremented. This is the sixth subsequent binary star evolved in the context of the commandline (all stars, or binary stars, evolved in a single execution of COMPAS are evolved in the context of the commandline), so the random seed is incremented from 12 to 18 (by 1 for each binary star evolved), and the value used for this binary star is 18.

Note that in this example, all binary stars were evolved using a unique random seed. This is because the values specified for the random seeds via the *--random-seed* option were 'well-behaved'. Unfortunately there is no reasonable way to protect the user against specifying duplicate random seeds – especially since the random see increments for each star or binary star. If the user chooses to specify multiple grid file lines with the same random seed, or initial random seeds that would collide with other random seed values and cause duplicates as they increment through ranges and sets, then there will be duplicate random seeds in the output files. Users should take care when specifying random seeds in grid files via the *--random-seed* option.

Running COMPAS via Python

A convenient method of managing the many program options provided by COMPAS is to run COMPAS via

Python, using a script to manage and specify the values of the program options.

Ranges and sets can be specified for options in the Python script file, but the range or set parameter must be enclosed in quotes – otherwise python tries to parse the construct. For example, to specify a set of metallicity values in the Python script file, use:

```
metallicity = 's[0.001,0.002,0.003,0.007,0.01,0.015,0.02]'
```

If the set parameter is not enclosed in quotes, Python will attempt to parse it, and will fail.

An example Python script is provided in the COMPAS suite: *pythonSubmit.py*. Users should copy this script and modify their copy to match their experimental requirements. Refer to the **Getting Started Guide** for more details.

COMPAS Output

Summary and status information during the evolution of stars is written to stdout; how much is written depends upon the value of the *--quiet* program option.

Detailed information is written to log files (described below). All COMPAS output files are created inside a container directory, specified by the *--output-container* program option.

If Detailed Output log files (see the *--detailed-output* program option) are created, they will be created inside a containing directory named 'Detailed_Output' within the COMPAS output container directory.

Also created in the COMPAS container directory is a file named 'Run_Details' in which COMPAS records some details of the run (COMPAS version, start time, program option values etc.). Note that the option values recorded in the Run details file are the values specified on the commandline, not the values specified in a grid file (if used).

COMPAS defines several standard log files that may be produced depending upon the simulation type (Single Star Evolution (SSE), or Binary Star Evolution (BSE), and the value of various program options. The standard log files are:

- Detailed Output log file
  Records detailed information for a star, or binary star, during evolution. Enable with program option `--detailed-output`.
- SwitchLog log file
  Records detailed information for all stars, or binary stars, at the time of each stellar type switch during evolution. Enable with program option `--switch-log`.
- Supernovae log file
  Records summary information for all stars that experience a SN event during evolution.
- System Parameters log file
  Records summary information for all stars, or binary stars, during evolution.
- Double Compact Objects log file
  Records summary information for all binary systems that form DCOs during BSE.
- Common Envelopes log file
  Records summary information for all binary systems that experience CEEs during BSE.
- Pulsar Evolution log file
  Records detailed Pulsar evolution information during BSE.
- RLOF file
  Records detailed information RLOF events during BSE. Enable with program option `--rlof-printing`.

Standard Log File Record Specifiers

Each standard log file has an associated log file record specifier that defines what data are to be written to the log files. Each record specifier is a list of known properties that are to be written as the log record for the log file associated with the record specifier. Default record specifiers for each of the standard log files are shown in Appendix **Default Log File Record Specifications**. The standard log file record specifiers can be defined by the user at run-time (see Section **Standard Log File Record Specification** below).

When specifying known properties, the property name must be prefixed with the property type. The current list of valid property types available for use is:

- STAR_PROPERTY
- STAR_1_PROPERTY
- STAR_2_PROPERTY
- SUPERNOVA_PROPERTY
- COMPANION_PROPERTY
- BINARY_PROPERTY
- PROGRAM_OPTION

The stellar property types (all types except BINARY_PROPERTY AND PROGRAM_OPTION) must be paired with properties from the stellar property list, the binary property type BINARY_PROPERTY with properties from the binary property list, and the program option type PROGRAM_OPTION with properties from the program option property list.

Standard Log File Record Specification

The standard log file record specifiers can be changed at run-time by supplying a definitions file via the *--logfile-definitions* program option.

The syntax of the definitions file is fairly simple. The definitions file is expected to contain zero or more log file record specifications, as explained below.

For the following specification:

::=	means ”expands to” or ”is defined as”
{ *x* }	means (possible) repetition: *x* may appear zero or more times
[*x*]	means x is optional: *x* may appear, or not
<name>	is a term (expression)
”abc”	means literal string ”abc”
|	means ”or”
#	indicates the start of a comment

Logfile Definitions File specification:

```
<def_file>     ::=  {<rec_spec>}

<rec_spec>     ::=  <rec_name> <op> ”{” { [ <props_list> ] } ” }” <spec_delim>

<rec_name>     ::=  ”SSE_SYSPARMS_REC”          |     # SSE only
                    ”SSE_DETAILED_REC”          |     # SSE only
                    ”SSE_SNE_REC”               |     # SSE only
                    ”SSE_SWITCH_REC”            |     # SSE only
                    ”BSE_SYSPARMS_REC”          |     # BSE only
                    ”BSE_SWITCH_REC”            |     # BSE only
                    ”BSE_DCO_REC”               |     # BSE only
                    ”BSE_SNE_REC”               |     # BSE only
                    ”BSE_CEE_REC”               |     # BSE only
                    ”BSE_PULSARS_REC”           |     # BSE only
                    ”BSE_RLOF_REC”              |     # BSE only
                    ”BSE_DETAILED_REC”          |     # BSE only

<op>           ::=  ”=” | ”+=” | ”-=”

<props_list>   ::=  <prop_spec> [ <props_delim> <props_list> ]

<prop_spec>    ::=  <prop_type> ”::” <prop_name> <prop_delim>

<spec_delim>   ::=  ”::” | ”EOL”

<prop_delim>   ::=  ”,” | <spec_delim>

<prop_type>    ::=  ”STAR_PROPERTY”             |     # SSE only
                    ”STAR1_PROPERTY”            |     # BSE only
                    ”STAR2_PROPERTY”            |     # BSE only
                    ”SUPERNOVA_PROPERTY”        |     # BSE only
                    ”COMPANION_PROPERTY”        |     # BSE only
                    ”BINARY_PROPERTY”           |     # BSE only
                    ”PROGRAM_OPTION”            |     # SSE or BSE

<prop_name>    ::=  valid property name for specified property type (see definitions in constants.h)
```

The file may contain comments. Comments are denoted by the hash/pound character ('#'). The hash character and any text following it on the line in which the hash character appears is ignored by the parser. The hash character can appear anywhere on a line - if it is the first character then the entire line is a comment and ignored by the parser, or it can follow valid symbols on a line, in which case the symbols before the hash character are parsed and interpreted by the parser.

A log file specification record is initially set to its default value (see Appendix E – Default Log File Record Specifications). The definitions file informs the code as to the modifications to the default values the user wants. This means that the definitions log file is not mandatory, and if the definitions file is not present, or contains no valid record specifiers, the log file record definitions will remain at their default values.

The assignment operator given in a record specification (<op> in the file specification above) can be one of "=", "+=", and "-=". The meanings of these are:

> "=" means that the record specifier should be assigned the list of properties specified in the braced-list following the "=" operator. The value of the record specifier prior to the assignment is discarded, and the new value set as described.
>
> "+=" means that the list of properties specified in the braced-list following the "+=" operator should be appended to the existing value of the record specifier. Note that the new properties are appended to the existing list, so will appear at the end of the list (properties are printed in the order they appear in the list).
>
> "-=" means that the list of properties specified in the braced-list following the "-=" operator should be subtracted from the existing value of the record specifier.

Example Log File Definitions File entries:

```
SSE_SYSPARMS_REC =  { STAR_PROPERTY::RANDOM_SEED
                      STAR_PROPERTY::RADIUS, STAR_PROPERTY::MASS,
                      STAR_PROPERTY::LUMINOSITY }

BSE_PULSARS_REC +=  { STAR_1_PROPERTY::LUMINOSITY,
                      STAR_2_PROPERTY::CORE_MASS,
                      BINARY_PROPERTY::SEMI_MAJOR_AXIS_RSOL,
                      COMPANION_PROPERTY::RADIUS }

BSE_PULSARS_REC -=  { SUPERNOVA_PROPERTY::TEMPERATURE }

BSE_PULSARS_REC +=  { PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_NS,
                      BINARY_PROPERTY::ORBITAL_VELOCITY }
```

A full example Log File Record Specifications File is shown in Appendix F – Example Log File Record Specifications File.

The record specifications in the definitions file are processed individually in the sequence they appear in the file, and are cumulative: for record specifications pertaining to the same record name, the output of earlier specifications is input to later specifications.

For each record specification:

- Properties requested to be added to an existing record specification that already exist in that record specification are ignored. Properties will not appear in a record specification twice.
- Properties requested to be subtracted from an existing record specification that do not exist in that record specification are ignored.

Note that neither of those circumstances will cause a parse error for the definitions file – in both cases the user's intent is satisfied.

Standard Log File Format

COMPAS can produce log files in several formats: Hierarchical Data Format version 5 (HDF5)[1], Comma Separated Values (CSV), Tab Separated Values (TSV), and Plain text: space separated values (TXT). The log file type is set using the *--logfile-type* program option.

Standard CSV, TSV, and TXT log files are human-readable files, and formatted in a similar fashion. Each standard CSV, TSV, and TXT log file consists of three header records followed by data records. Header records and data records are delimiter separated fields, and the fields as specified by the log file record specifier.

The header records for all standard CSV, TSV, and TXT log files are:

Header record 1: Column Data Type Names

Header record 2: Column Units (where applicable)

Header record 3: Column Headings

Column Data Type Names are taken from the set { BOOL, INT, FLOAT, STRING }, where

BOOL	the data value will be a boolean value. Boolean data values will be recorded in the log file in either numerical format (1 or 0, where 1 = TRUE and 0 = FALSE), or string format ('TRUE' or 'FALSE'), depending upon the value of the *--print-bool-as-string* program option.
INT	the data value will be an integer number.
FLOAT	the data value will be a floating-point number.
STRING	the data value will be a text string.

Column Units is a string indicating the units of the corresponding data values (e.g. 'Msol*AU∧2*yr∧-1', 'Msol', 'AU', etc.). The Column Units value may be blank where units are not applicable, or may be one of:

'Count'	the data value is the total of a counted entity.
'State'	the data value describes a state (e.g. 'Unbound' state is 'TRUE' or 'FALSE').
'Event'	the data value describes an event status (e.g. 'Simultaneous_RLOF' is 'TRUE').

Column Headings are string labels that describe the corresponding data values. The heading strings for stellar properties of constituent stars of a binary will have appropriate identifiers appended. That is, heading strings for:

STAR_1_PROPERTY::*properties* will have '**(1)**' appended

STAR_2_PROPERTY::*properties* will have '**(2)**' appended

SUPERNOVA_PROPERTY::*properties* will have '**(SN)**' appended

COMPANION_PROPERTY::*properties* will have '**(CP)**' appended

HDF5 files are not human-readable. The HDF5 file format supports large, complex, heterogeneous data, enabling the data to be stored in a structured way in a single file. When the HDF5 format is specified for

[1] `https://www.hdfgroup.org/`

COMPAS log files, a single HDF5 file is produced for non-detailed output log files, containing all non-detailed output log files described above. Detailed output files are created, as for other logfile types, as individual files (in this case, HDF5 files), in the 'Detailed_Output' container directory.

Each file described above is created as a "group" within the HDF5 file, with the name of the group set to the name of the file (e.g. "BSE_System_Parameters). Each column in the files described above is created as a "dataset" within its corresponding group in the HDF5 file, with the name of the datset set to the column header as described above (e.g. "Mass(1)"). Each dataset in an HDF5 file is typed, and the dataset data types are set to the column data types as described above. The column units described above are attached to their corresponding datasets in the HDF5 file as "attributes".

Developer Guide

TeamCOMPAS welcomes the active involvement of colleagues and others interested in the ongoing development and improvement of the COMPAS software. We hope this Developer Guide helps anyone interested in contributing to the COMPAS software. We expect this guide to be a living document and improve along with the improvements made to the software.

Single Star Evolution

Class Hierarchy

The SSE architecture is based on the classification of individual stars, with each stellar classification being described by a separate C++ class. Figure 1 shows the SSE class diagram, where the arrows indicate inheritance. The COMPAS C++ code is implemented using multiple inheritance, and all stellar classes also inherit directly from the Base Star class (arrows not shown in Figure 1 for clarity). Each of the stellar classes encapsulates data structures and algorithms specific to the evolutionary phase corresponding to the class.

The main class for single star evolution is the **Star** class. The Star class is a wrapper that abstracts away the details of the star and the evolution. Internally the Star class maintains a pointer to an object representing the star being evolved, with that object being an instance of one of the following classes:

MS_lte_07
MS_gt_07
CH
HG
FGB
CHeB
EAGB
TPAGB
HeMS
HeHG
HeGB
HeWD
COWD
ONeWD
NS
BH
MR

which track the phases from Hurley et al. (2000), with the exception of the **CH** class for Chemically Homogeneous stars, which is not described in Hurley et al. (2000).

Several other SSE classes are defined:

BaseStar
MainSequence
GiantBranch
Remnants
WhiteDwarfs

These extra classes are included to allow inheritance of common functionality.

The **BaseStar** class is the main class for the underlying star object held by the Star class. The BaseStar class defines all member variables, and many member functions that provide common functionality. Similarly, the **MainSequence** and **GiantBranch** classes provide repositories for common functionality for main sequence and giant branch stars respectively, and the the **Remnants** and **WhiteDwarfs** classes provide repositories for common functionality for remnant and white dwarf stars respectively.

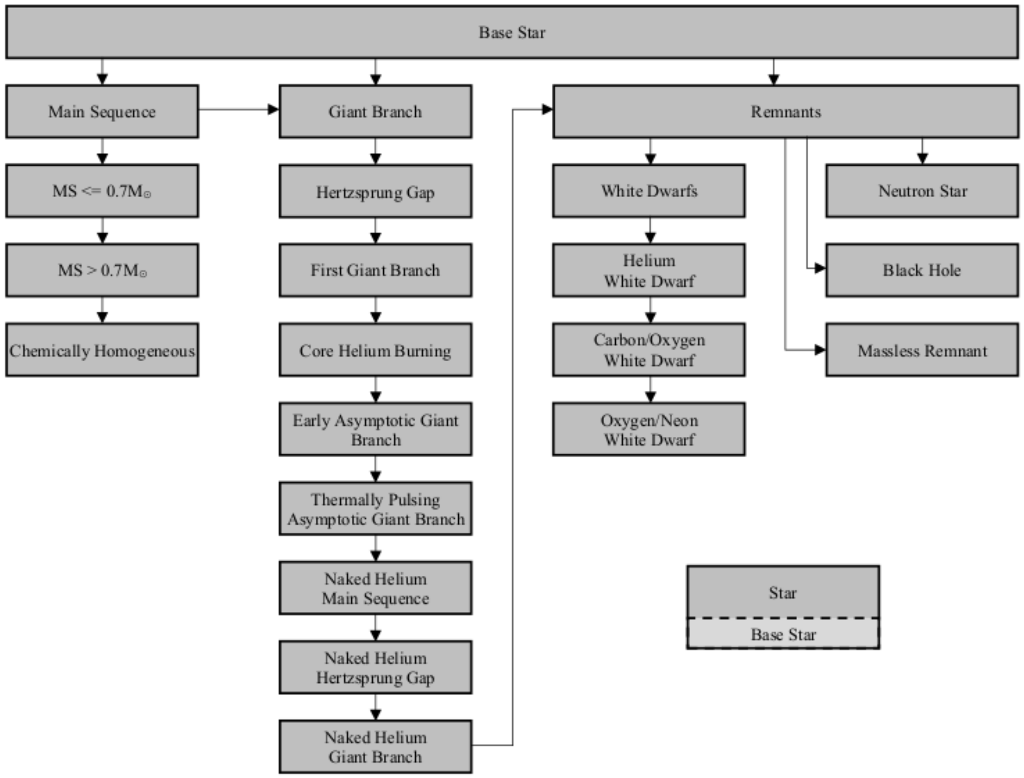

Figure 1: SSE class diagram.

CH (Chemically Homogeneous) class stars inherit from the **MS_gt_07** class because (in this implementation) they are just (large) main sequence stars that have a static radius.

HG (Hertzsprung Gap) class stars inherit from the GiantBranch class because they share the giant branch parameters described in Hurley et al. (2000), section 5.2.

Each class has its own set of member functions that calculate various attributes of the star according to the phase the class represents (using the equations and parameters from Hurley et al. (2000) where applicable).

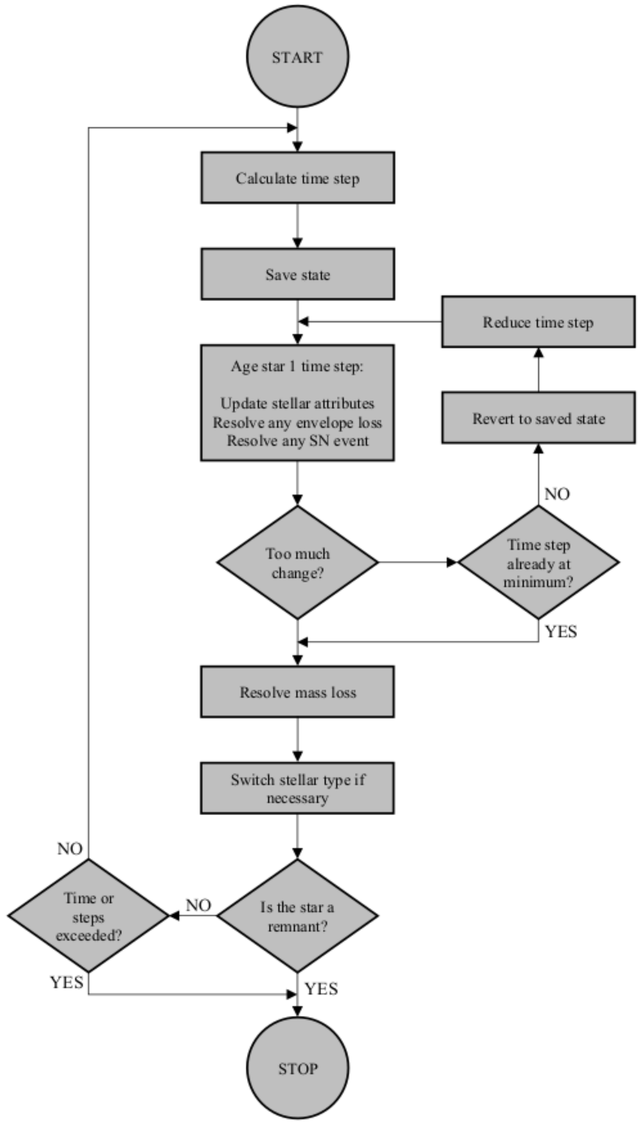

Figure 2: High-level SSE evolution.

Evolution Model

The high-level stellar evolution model is shown in Figure 2.

The stellar evolution model is driven by the **Evolve()** function in the Star class, which evolves the star through its entire lifetime by doing the following:

DO:

1. calculate time step
    (a) calculate the giant branch parameters (as necessary)
    (b) calculate the timescales
    (c) choose time step
2. save the state of the underlying star object
3. DO:
    (a) evolve a single time step
    (b) if too much change
        i. revert to the saved state
        ii. reduce the size of the time step

    UNTIL timestep not reduced
4. resolve any mass loss
    (a) update initial mass (mass0)
    (b) update age after mass loss
    (c) apply mass transfer rejuvenation factor
5. evolve to the next stellar type if necessary

WHILE the underlying star object is not one of: { HeWD, COWD, ONeWD, NS, BH, MR }

Evolving the star through a single time step (step 3a above) is driven by the **UpdateAttributesAndAgeOneTimestep()** function in the BaseStar class which does the following:

1. check if the star should be a massless remnant
2. check if the star is a supernova

if evolution on the phase should be performed

3. evolve the star on the phase – update stellar attributes
4. check if the star should evolve off the current phase to a different stellar type

else

5. ready the star for the next time step

Evolving the star on its current phase, and off the current phase and preparing to evolve to a different stellar type, is handled by two functions in the BaseStar class: **EvolveOnPhase()** and **ResolveEndOfPhase()**.

The EvolveOnPhase() function does the following:

1. Calculate Tau
2. Calculate CO Core Mass
3. Calculate Core Mass
4. Calculate He Core Mass
5. Calculate Luminosity
6. Calculate Radius
7. Calculate Perturbation Mu
8. Perturb Luminosity and Radius
9. Calculate Temperature
10. Resolve possible envelope loss

Each of the calculations in the EvolveOnPhase() function is performed in the context of the star evolving on its current phase. Each of the classes implements their own version of the calculations (via member functions) – some may inherit functions from the inheritance chain, while others might just return the value unchanged if the calculation is not relevant to their stellar type.

The ResolveEndOfPhase() function does the following:

1. Resolve possible envelope loss
2. Calculate Tau
3. Calculate CO Core Mass
4. Calculate Core Mass
5. Calculate He Core Mass
6. Calculate Luminosity
7. Calculate Radius
8. Calculate Perturbation Mu
9. Perturb Luminosity and Radius
10. Calculate Temperature
11. Evolve star to next phase

Each of the calculations in the ResolveEndOfPhase() function is performed in the context of the star evolving off its current phase to the next phase.

The remainder of the code (in general terms) supports these main driver functions.

Binary Star Evolution

Class Hierarchy

The BSE inheritance hierarchy is as follows:

BinaryStar → BaseBinaryStar

(Star →) BinaryConstituentStar (star1)
(Star →) BinaryConstituentStar (star2)

The main class for binary star evolution is the **BinaryStar** class. The BinaryStar class is a wrapper that abstracts away the details of the binary star and the evolution. Internally the BinaryStar class maintains a pointer to an object representing the binary star being evolved, with that object being an instance of the **BaseBinaryStar** class.

The **BaseBinaryStar** class is a container class for the objects that represent the component stars of a binary system. An instance of the BaseBinaryStar class is a binary system being evolved by COMPAS, and contains a **BinaryConstituentStar** class object for each of the component stars (i.e. the primary and secondary stars), as well as data structures and algorithms specific to the evolution of a binary system.

The **BinaryConstituentStar** class is a wrapper class for the SSE Star class. An instance of the BinaryConstituentStar class is a single component star of a binary system being evolved by COMPAS, and contains a Star class object that will evolve over time through various SSE classes shown in Figure 1. The BinaryConstituentStar class defines additional data structures and algorithms (to the data structures and algorithms provided by the SSE classes) required to support the evolution of a binary system component star.

The BaseBinaryStar class is the main class for the underlying binary star object held by the BinaryStar class. The BaseBinaryStar class defines all member variables that pertain specifically to a binary star, and many member functions that provide binary-star specific functionality. Internally, the BaseBinaryStar class maintains pointers to the two **BinaryConstituentStar** class objects that constitute the binary star.

The BinaryConstituentStar class inherits from the Star class, so objects instantiated from the BinaryConstituentStar class inherit the characteristics of the Star class, particularly the stellar evolution model. The BinaryConstituentStar class defines member variables and functions that pertain specifically to a constituent star of a binary system but that do not (generally) pertain to single stars that are not part of a binary system (there are some functions that are defined in the BaseStar class and its derived classes that deal with binary star attributes and behaviour – in some cases the stellar attributes that are required to make these calculations reside in the BaseStar class so it is easier and cleaner to define the functions there).

Evolution Model

The high-level binary evolution model is shown in Figure 3.

The binary evolution model is driven by the **Evolve()** function in the BaseBinaryStar class, which evolves the star through its entire lifetime by doing the following:

```
if touching
        STOP = true
else
        calculate initial time step
        STOP = false

DO WHILE NOT STOP AND NOT max iterations:

        evolve a single time step
            evolve each constituent star a single time step (see SSE evolution)

        if error OR unbound OR touching OR Massless Remnant
            STOP = true
        else
            evaluate the binary

                calculate mass transfer
                calculate winds mass loss

                if common envelope
                    resolve common envelope
                else if supernova
                    resolve supernova
                else
                    resolve mass changes

                evaluate supernovae

                calculate total energy and angular momentum
                update magnetic field and spin: both constituent stars

            if unbound OR touching OR merger
                STOP = true
            else
                if NS+BH
                    resolve coalescence

                    STOP = true
                else
                    if WD+WD OR max time
                        STOP = true
                    else
                        if NOT max iterations
                            calculate new time step
```

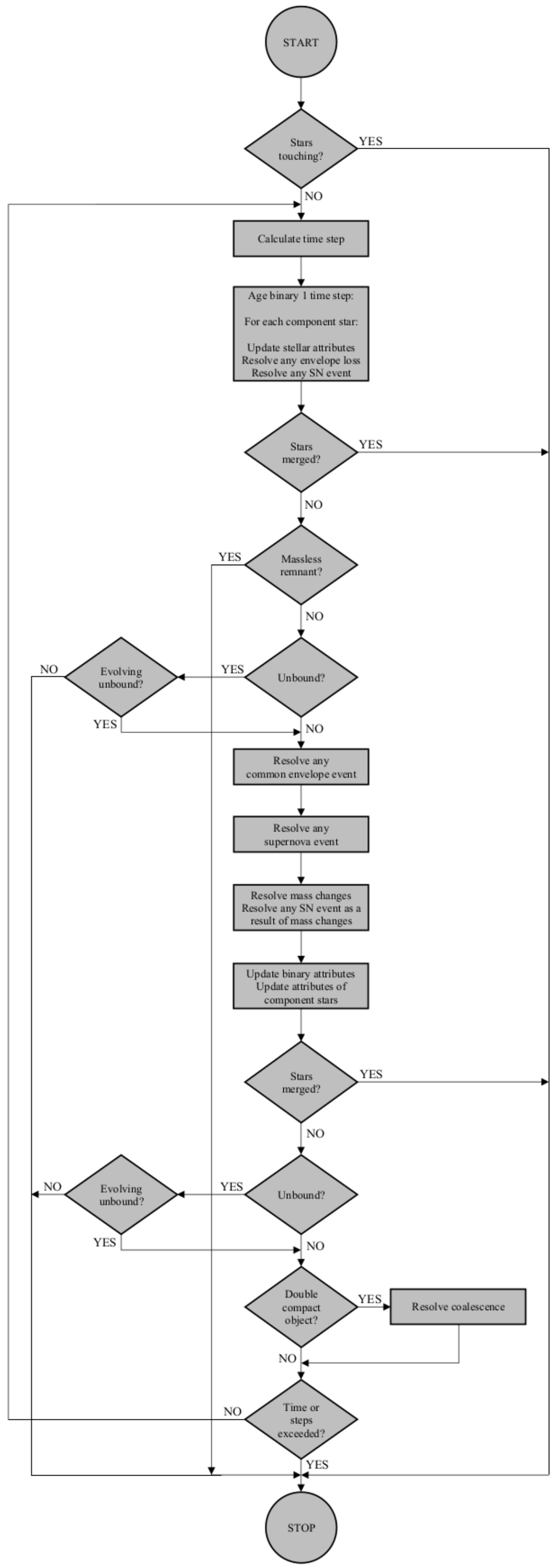

Figure 3: High-level BSE evolution.

Object Identifiers

All objects (instantiations of a class) are assigned unique object identifiers of type OBJECT_ID (unsigned long int - see **constants.h** for the typedef). The purpose of the unique object id is to aid in object tracking and debugging.

Note that the object id is not the same as, nor does it supersede, the RANDOM SEED value assigned to each single or binary star. The RANDOM SEED is used to seed the random number generator, and can be used to uniquely identify a single or binary star. The object id is more granular the the RANDOM SEED. Each binary star is comprised of multiple objects: the BinaryStar object, which contains two BaseBinaryStar objects (the object undergoing evolution, and a saved copy); each BaseBinaryStar object contains two BinaryConstituentStar objects (one for each of the constituent stars), and each BinaryConstituentStar object inherits from the Star class, which contains two BaseStar objects (the underlying star and a saved copy). Whereas the RANDOM SEED uniquely identifies (for example) a binary star, and so identifies the collection of objects that comprise the binary star, the object ids uniquely identify the constituent objects of the binary star.

As well as unique object ids, all objects are assigned an object type (of type OBJECT_TYPE – see **constants.h** for the enum class declaring OBJECT_TYPE), and a stellar type where applicable (of type STELLAR_TYPE – see **constants.h** for the enum class declaring STELLAR_TYPE).

Objects should expose the following functions:

```
OBJECT_ID ObjectId() const { return m_ObjectId; }
OBJECT_TYPE ObjectType() const { return m_ObjectType; }
STELLAR_TYPE StellarType() const { return m_StellarType; }
```

If any of the functions are not applicable to the object, then they must return "*::NONE" (all objects should implement ObjectId() correctly).

Any object that uses the Errors service (i.e. the SHOW_* macros) ***must*** expose these functions: the functions are called by the SHOW_* macros (the Errors service is described later in this document).

Services

A number of services have been provided to help simplify the code. These are:

- Program Options
- Random Numbers
- Logging and Debugging
- Error Handling

The code for each service is encapsulated in a singleton object (an instantiation of the relevant class). The singleton design pattern allows the definition of a class that can only be instantiated once, and that instance effectively exists as a global object available to all the code without having to be passed around as a parameter. Singletons are a little anti-OO, but provided they are used judiciously are not necessarily a bad thing, and can be very useful in certain circumstances.

Program Options

A Program Options service is provided encapsulated in a singleton object (an instantiation of the Options class).

The Options class member variables are private, and public getter functions have been created for the program options currently used in the code.

The Options service can be accessed by referring to the Options::Instance() object. For example, to retrieve the value of the `--`*quiet* program option, call the Quiet() getter function:

bool quiet = Options::Instance()→Quiet();

Since that could become unwieldy, there is a convenience macro to access the Options service. The macro just defines "OPTIONS" as "Options::Instance()", so retrieving the value of the `--`*quiet* program option can be written as:

bool quiet = OPTIONS→Quiet();

The Options service must be initialised before use. Initialise the Options service by calling the Initialise() function:

COMMANDLINE_STATUS programStatus = OPTIONS→Initialise(argc, argv);

(see **constants.h** for details of the COMMANDLINE_STATUS type)

Random Numbers

A Random Number service is provided, with the gsl Random Number Generator encapsulated in a singleton object (an instantiation of the Rand class).

The Rand class member variables are private, and public functions have been created for random number functionality required by the code.

The Rand service can be accessed by referring to the Rand::Instance() object. For example, to generate a uniform random floating point number in the range [0, 1), call the Random() function:

double u = Rand::Instance()→Random();

Since that could become unwieldy, there is a convenience macro to access the Rand service. The macro just defines "RAND" as "Rand::Instance()", so calling the Random() function can be written as:

double u = RAND→Random();

The Rand service must be initialised before use. Initialise the Rand service by calling the Initialise() function:

RAND→Initialise();

Dynamically allocated memory associated with the gsl random number generator should be returned to the system by calling the Free() function:

RAND→Free();

before exiting the program.

The Rand service provides the following public member functions:

void Initialise()
Initialises the gsl random number generator. If the environment variable GSL_RNG_SEED exists, the gsl random number generator is seeded with the value of the environment variable, otherwise it is seeded with the current time.

void Free()
Frees any dynamically allocated memory.

unsigned long int Seed(const unsigned long p_Seed)
Sets the seed for the gsl random number generator to p_Seed. The return value is the seed.

unsigned long int DefaultSeed()
Returns the gsl default seed (gsl_rng_default_seed)

double Random(void)
Returns a random floating point number uniformly distributed in the range [0.0, 1.0)

double Random(const double p_Lower, const double p_Upper)
Returns a random floating point number uniformly distributed in the range [p_Lower, p_Upper), where p_Lower ≤ p_Upper.
(p_Lower and p_Upper will be swapped if p_Lower > p_Upper as passed)

double RandomGaussian(const double p_Sigma)
Returns a Gaussian random variate, with mean 0.0 and standard deviation p_Sigma

int RandomInt(const int p_Lower, const int p_Upper)
Returns a random integer number uniformly distributed in the range [p_Lower, p_Upper), where p_Lower ≤ p_Upper.
(p_Lower and p_Upper will be swapped if p_Lower > p_Upper as passed)

int RandomInt(const int p_Upper) Returns a random integer number uniformly distributed in the range [0, p_Upper),where 0 ≤ p_Upper. Returns 0 if p_Upper < 0.

Logging & Debugging

A logging and debugging service is provided encapsulated in a singleton object (an instantiation of the Log class).

The logging functionality was first implemented when the Single Star Evolution code was refactored, and the base-level of logging was sufficient for the needs of the SSE code. Refactoring the Binary Star Evolution code highlighted the need for expanded logging functionality. To provide for the logging needs of the BSE code, new functionality was added almost as a wrapper around the original, base-level logging functionality. Some of the original base-level logging functionality has almost been rendered redundant by the new functionality implemented for BSE code, but it remains (almost) in its entirety because it may still be useful in some circumstances.

When the base-level logging functionality was created, debugging functionality was also provided, as well as a set of macros to make debugging and the issuing of warning messages easier. A set of logging macros was also provided to make logging easier. The debug macros are still useful, and their use is encouraged (rather than inserting print statements using std::cout or std::cerr).

When the BSE code was refactored, some rudimentary error handling functionality was also provided in the form of the Errors service - an attempt at making error handling easier. Some of the functionality provided by the Errors service supersedes the DBG_WARN* macros provided as part of the Log class, but the DBG_WARN* macros are still useful in some circumstances (and in fact are still used in various places in the code). The LOG* macros are somewhat less useful, but remain in case the original base-level logging functionality (that which underlies the expanded logging functionality) is used in the future (as mentioned above, it could still be useful in some circumstances). The Errors service is described in Section **Error Handling**.

The expanded logging functionality introduces Standard Log Files - described in Section **Extended Logging**.

Base-Level Logging

The Log class member variables are private, and public functions have been created for logging and debugging functionality required by the code.

The Log service can be accessed by referring to the Log::Instance() object. For example, to stop the logging service, call the Stop() function:

    Log::Instance()→Stop();

Since that could become unwieldy, there is a convenience macro to access the Log service. The macro just defines “LOGGING” as “Log::Instance()”, so calling the Stop() function can be written as:

    LOGGING→Stop();

The Log service must be initialised before logging and debugging functionality can be used. Initialise logging by calling the Start() function:

LOGGING→Start(...)

Refer to the description of the **Start()** function below for parameter definitions.

The Log service should be stopped before exiting the program – this ensures all open log files are flushed to disk and closed properly. Stop logging by calling the Stop() function:

LOGGING→Stop(...)

Refer to the description of the **Stop()** function below for parameter definitions.

The Log service provides the following public member functions:

VOID Start(

outputPath,	- STRING,	the name of the top-level directory in which log files will be created.
containerName,	- STRING,	the name of the directory to be created at *outputPath* to hold all log files.
logfilePrefix,	- STRING,	prefix for logfile names (can be blank).
logLevel,	- INT,	logging level (see below).
logClasses,	- STRING[],	enabled logging classes (see below).
debugLevel,	- INT,	debug level (see below).
debugClasses,	- STRING[],	enabled debug classes (see below).
debugToLogfile,	- BOOL,	flag indicating whether debug statements should also be written to log file.
errorsToLogfile,	- BOOL,	flag indicating whether error messages should also be written to log file.
delimiter	- STRING,	the default field delimiter in log file records. Typically a single character.

)

Initialises the logging and debugging service. Logging parameters are set per the program options specified (using default values if no options are specified by the user). The log file container directory is created. If a directory with the name as given by the *containerName* parameter already exists, a version number will be appended to the directory name. The **Run_Details** file is created within the logfile container directory. Log files to which debug statements and error messages will be created and opened if required.

This function does not return a value.

VOID Stop(

objectStats	- tuple<INT, INT>,	number of objects (stars or binaries) requested, count created.

)

Stops the logging and debugging service. All open log files are flushed to disk and closed (including and Standard Log Files open - see description of Standard Log Files in Section **Extended Logging**). The Run_Details file is populated and closed.

This function does not return a value.

BOOL Enabled()

Returns a boolean indicating whether the Log service is enabled – true indicates the Log service is enable and available; false indicates the Log service is not enable and so not available.

INT Open(

logFileName,	- STRING,	the name of the log file to be created and opened. This should be the filename only – the path, prefix and extensions are added by the logging service. If the file already exists, the logging service will append a version number to the name if necessary (see *append* parameter below).
append,	- BOOL,	flag indicating whether the file should be opened in append mode (i.e. existing data is preserved) and new records written to the file appended, or whether a new file should be opened (with version number if necessary).
timeStamps,	- BOOL,	flag indicating whether timestamps should be written with each log file record.
labels,	- BOOL,	flag indicating whether a label should be written with each log record. This is useful when different types of logging data are being written to the same log file file.
delimiter	- STRING,	(optional) the field delimiter for this log file. If not provided the default delimiter is used (see *Start()*). Typically a single character.

)

Opens a log file. If the append parameter is true and a file name *logFilename* exists, the existing file will be opened and the existing contents retained, otherwise a new file will be created and opened (not a Standard Log File – see description of Standard Log Files later in Section **Extended Logging**).

The log file container directory is created at the path specified by the *outputPath* parameter passed to the Start() function. New log files are created in the logfile container directory. BSE Detailed log files are created in the Detailed_Output directory, which is created in the log file container directory if required.

The filename is prefixed by the *logfilePrefix* parameter passed to the Start() function.

The file extension is based on the *delimiter* parameter passed to the Start() function:

SPACE	will result in a file extension of ".txt"
TAB	will result in a file extension of ".tsv"
COMMA	will result in a file extension of ".csv"

If a file with the name as given by the *logFilename* parameter already exists, and the *append* parameter is false, a version number will be appended to the filename before the extension (this functionality is largely redundant since the implementation of the log file container directory).

The integer log file identifier is returned to the caller. A value of -1 indicates the log file was not opened successfully. Multiple log files can be open simultaneously – referenced by the identifier returned.

BOOL Close(

logFileId	- INT,	the identifier of the log file to be closed (as returned by *Open()*).

)

Closes the log file specified by the *logFileId* parameter. If the log file specified by the *logFileId* parameter is open, it is flushed to disk and closed.

The function returns a boolean indicating whether the file was closed successfully.

BOOL Write(

logFileId,	- INT,	the identifier of the log file to be written.
logClass,	- STRING,	the log class of the record to be written. An empty string ("") satisfies all checks against enabled classes.
logLevel,	- INT,	the log level of the record to be written. *logLevel = 0* satisfies all checks against enabled levels.
logString	- STRING,	the string to be written to the log file.

)

Writes an unformatted record to the specified log file. If the Log service is enabled, the specified log file is active, and the log class and log level passed are enabled (see discussion of log classes and levels), the string is written to the file.

The function returns a boolean indicating whether the record was written successfully. If an error occurred the log file will be disabled.

BOOL Put(

logFileId,	- INT,	the identifier of the log file to be written.
logClass,	- STRING,	the log class of the record to be written. An empty string ("") satisfies all checks against enabled classes.
logLevel,	- INT,	the log level of the record to be written. *logLevel = 0* satisfies all checks against enabled levels.
logString	- STRING,	the string to be written to the log file.

)

Writes a minimally formatted record to the specified log file. If the Log service is enabled, the specified log file is active, and the log class and log level passed are enabled (see discussion of log classes and levels), the string is written to the file.

If labels are enabled for the log file, a label will be prepended to the record. The label text will be the *logClass* parameter.

If timestamps are enabled for the log file, a formatted timestamp is prepended to the record. The timestamp format is *yyyymmdd hh:mm:ss*.

The function returns a boolean indicating whether the record was written successfully. If an error occurred the log file will be disabled.

BOOL Debug(

debugClass,	- STRING,	the log class of the record to be written. An empty string ("") satisfies all checks against enabled classes.
debugLevel,	- INT,	the log level of the record to be written. *logLevel = 0* satisfies all checks against enabled levels.
debugString	- STRING,	the string to be written to stdout (and optionally to file).

)

Writes *debugString* to stdout and, if logging is active and so configured (via program option --*debug-to-file*), writes *debugString* to the debug log file.

The function returns a boolean indicating whether the record was written successfully. If an error occurred writing to the debug log file, the log file will be disabled.

BOOL DebugWait(

debugClass,	- STRING,	the log class of the record to be written. An empty string ("") satisfies all checks against enabled classes.
debugLevel,	- INT,	the log level of the record to be written. *logLevel = 0* satisfies all checks against enabled levels.
debugString	- STRING,	the string to be written to stdout (and optionally to file).

)

Writes *debugString* to stdout and, if logging is active and so configured (via program option *--debug-to-file*), writes *debugString* to the debug log file, then waits for user input.

The function returns a boolean indicating whether the record was written successfully. If an error occurred writing to the debug log file, the log file will be disabled.

BOOL Error(

errorString	- STRING,	the string to be written to stdout (and optionally to file).

)

Writes *errorString* to stdout and, if logging is active and so configured (via program option *--errors-to-file*), writes *errorString* to the error log file, then waits for user input.

The function returns a boolean indicating whether the record was written successfully. If an error occurred writing to the error log file, the log file will be disabled.

VOID Squawk(

squawkString	- STRING,	the string to be written to stderr.

)

Writes *squawkString* to stderr.

VOID Say(

sayClass,	- STRING,	the log class of the record to be written. An empty string ("") satisfies all checks against enabled classes.
sayLevel,	- INT,	the log level of the record to be written. *logLevel = 0* satisfies all checks against enabled levels.
sayString	- STRING,	the string to be written to stdout.

)

Writes *sayString* to stdout.

The filename to which debug records are written when Start() parameter *debugToLogfile* is true is declared in **constants.h** – see the LOGFILE enum class and associated descriptor map LOGFILE_DESCRIPTOR. Currently the name is "Debug_Log".

Extended Logging

The Logging service was extended to support standard log files for Binary Star Evolution (SSE also uses the extended logging). The standard log files defined are:

- SSE Parameters log file
- BSE System Parameters log file
- BSE Detailed Output log file
- BSE Double Compact Objects log file
- BSE Common Envelopes log file
- BSE Supernovae log file
- BSE Pulsar Evolution log file

The Logging service maintains information about each of the standard log files, and will handle creating, opening, writing and closing the files. For each execution of the COMPAS program that evolves binary stars, one (and only one) of each of the log file listed above will be created, except for the Detailed Output log in which case there will be one log file created for each binary star evolved.

The Logging service provides the following public member functions specifically for managing standard log files:

void LogSingleStarParameters(Star, Id**)**

void LogBinarySystemParameters(Binary**)**

void LogDetailedOutput(Binary, Id**)**

void LogDoubleCompactObject(Binary**)**

void LogCommonEnvelope(Binary**)**

void LogSupernovaDetails(Binary**)**

void LogPulsarEvolutionParameters(Binary**)**

Each of the BSE functions is passed a pointer to the binary star for which details are to be logged, and in the case of the Detailed Output log file, an integer identifier (typically the loop index of the binary star) that is appended to the log file name.

The SSE function is passed a pointer to the single star for which details are to be logged, and an integer identifier (typically the loop index of the star) that is appended to the log file name.

Each of the functions listed above will, if necessary, create and open the appropriate log file. Internally the Log service opens (creates first if necessary) once at first use, and keeps the files open for the life of the program.

The Log service provides a further two functions to manage standard log files:

BOOL CloseStandardFile(LogFile)

Flushes and closes the specified standard log file. The function returns a boolean indicating whether the log file was closed successfully.

BOOL CloseAllStandardFiles()

Flushes and closes all currently open standard log files. The function returns a boolean indicating whether all standard log files were closed successfully.

Standard log file names are supplied via program options, with default values declared in **constants.h**.

Logging & Debugging Macros

Logging Macros

LOG(id, . . .)

Writes a log record to the log file specified by *id*. Use:

LOG(id, string)	- writes *string* to the log file.
LOG(id, level, string)	- writes *string* to the log file if *level* is ≤ *id* in **Start()**.
LOG(id, class, level, string)	- writes *string* to the log file if *class* is in *logClasses* in **Start()** and *level* is ≤ *id* in **Start()**.

Default *class* is ””; default *level* is 0

Examples:

```
LOG(SSEfileId, ”This is a log record”);
LOG(OutputFile2Id, ”The value of x is ” << x << ” km”);
LOG(MyLogfileId, 2, ”Log string”);
LOG(SSEfileId, ”CHeB”, 4, ”This is a CHeB only log record”);
```

LOG_ID(id, . . .)

Writes a log record prepended with calling function name to the log file specified by *id*. Use:

LOG_ID(id)	- writes the name of calling function to the log file.
LOG_ID(id, string)	- writes *string* prepended with name of calling function to the log file.
LOG_ID(id, level, string)	- writes *string* prepended with name of calling function to the log file if *level* is ≤ *logLevel* in **Start()**.
LOG_ID(id, class, level, string)	- writes *string* prepended with name of calling function to the log file if *class* is in *logClasses* in **Start()** and *level* is ≤ *logLevel* in **Start()**.

Default *class* is ””; default *level* is 0

Examples:

```
LOG_ID(Outf1Id);
LOG_ID(Outf2Id, ”This is a log record”);
LOG_ID(MyLogfileId, ”The value of x is ” << x << ” km”);
LOG_ID(OutputFile2Id, 2, ”Log string”);
LOG_ID(CHeBfileId, ”CHeB”, 4, ”This is a CHeB only log record”);
```

LOG_IF(id, cond, . . .)

Writes a log record to the log file specified by *id* if the condition given by *cond* is met. Use:

LOG_IF(id, cond, string)	- writes *string* to the log file if *cond* is true.
LOG_IF(id, cond, level, string)	- writes *string* to the log file if *cond* is true and if *level* is ≤ *logLevel* in **Start()**.
LOG_IF(id, cond, class, level, string)	- writes *string* to the log file if *cond* is true, *class* is in *logClasses* in **Start()**, and *level* is ≤ *logLevel* in **Start()**.

Default *class* is ””; default *level* is 0

Examples:

LOG_IF(MyLogfileId, a > 1.0, ”This is a log record”);
LOG_IF(SSEfileId, (b == c && a > x), ”The value of x is ” << x << ” km”);
LOG_IF(CHeBfileId, flag, 2, ”Log string”);
LOG_IF(SSEfileId, (x >= y), ”CHeB”, 4, ”This is a CHeB only log record”);

LOG_ID_IF(id, . . .)

Writes a log record prepended with calling function name to the log file specified by *id* if the condition given by *cond* is met. Use: see **LOG_ID(id, . . .)** and **LOG_IF(id, cond, . . .)** above.

The logging macros described above are also provided in a verbose variant. The verbose macros function the same way as their non-verbose counterparts, with the added functionality that the log records written to the log file will be written to stdout as well. The verbose logging macros are:

LOGV(id, . . .)
LOGV_ID(id, . . .)
LOGV_IF(id, cond, . . .)
LOGV_ID_IF(id, cond, . . .)

A further four macros are provided that allow writing directly to stdout rather than a log file. These are:

SAY(. . .)
SAY_ID(. . .)
SAY_IF(cond, . . .)
SAY_ID_IF(cond, . . .)

The SAY macros function the same way as their LOG counterparts, but write directly to stdout instead of a log file. The SAY macros honour the logging classes and level.

Debugging Macros

A set of macros similar to the logging macros is also provided for debugging purposes.

The debugging macros write directly to stdout rather than the log file, but their output can also be written to the log file if desired (see the *debugToLogfile* parameter of Start(), and the --*debug-to-file* program option described above).

A major difference between the logging macros and the debugging macros is that the debugging macros can be defined away. The debugging macro definitions are enclosed in an #ifdef enclosure, and are only present in the source code if #DEBUG is defined. This means that if #DEBUG is not defined (#undef), all debugging statements using the debugging macros will be removed from the source code by the preprocessor before the source is compiled. Un-defining #DEBUG not only prevents bloat of unused code in the executable, it improves performance. Many of the functions in the code are called hundreds of thousands, if not millions, of times as the stellar evolution proceeds. Even if the debugging classes and debugging level are set so that no debug statement is displayed, just checking the debugging level every time a function is called increases the run time of the program. The suggested use is to enable the debugging macros (#define DEBUG) while developing new code, and disable them (#undef DEBUG) to produce a production version of the executable.

The debugging macros provided are:

DBG(. . .)	analogous to the **LOG(. . .)** macro
DBG_ID(. . .)	analogous to the **LOG_ID(. . .)** macro
DBG_IF(cond, . . .)	analogous to the **LOG_IF(. . .)** macro
DBG_ID_IF(cond, . . .)	analogous to the **LOG_ID_IF(. . .)** macro

Two further debugging macros are provided:

DBG_WAIT(. . .)
DBG_WAIT_IF(cond, . . .)

The **DBG_WAIT** macros function in the same way as their non-wait counterparts (**DBG(. . .)** and **DBG_IF(cond, . . .)**), with the added functionality that they will pause execution of the program and wait for user input before proceeding.

A set of macros for printing warning message is also provided. These are the **DBG_WARN** macros:

DBG_WARN(...)	analogous to the LOG(. . .) macro
DBG_WARN_ID(...)	analogous to the LOG_ID(. . .) macro
DBG_WARN_IF(...)	analogous to the LOG_IF(. . .) macro
DBG_WARN_ID_IF(...)	analogous to the LOG_ID_IF(. . .) macro

The **DBG_WARN** macros write to stdout via the **SAY** macro, so honour the logging classes and level, and are not written to the debug or errors files.

Note that the *id* parameter of the **LOG** macros (to specify the logfileId) is not required for the **DBG** macros (the filename to which debug records are written is declared in **constants.h** – see the LOGFILE enum class and associate descriptor map LOGFILE_DESCRIPTOR).

Error Handling

An error handling service is provided encapsulated in a singleton object (an instantiation of the Errors class).

The Errors service provides global error handling functionality. Following is a brief description of the Errors service (full documentation coming soon...):

Errors are defined in the error catalog in constants.h (see ERROR_CATALOG). It could be useful to move the catalog to a file so it can be changed without changing the code, or even have multiple catalogs provided for internationalisation – a task for later.

Errors defined in the error catalog have a scope and message text. The scope is used to determine when/if an error should be printed.

The current values for scope are:

NEVER	the error will not be printed.
ALWAYS	the error will always be printed.
FIRST	the error will be printed only on the first time it is encountered anywhere in the program.
FIRST_IN_OBJECT_TYPE	the error will be printed only on the first time it is encountered anywhere in objects of the same type (e.g. Binary Star objects).
FIRST_IN_STELLAR_TYPE	the error will be printed only on the first time it is encountered anywhere in objects of the same stellar type (e.g. HeWD Star objects).
FIRST_IN_OBJECT_ID	the error will be printed only on the first time it is encountered anywhere in an object instance.
FIRST_IN_FUNCTION	the error will be printed only on the first time it is encountered anywhere in the same function of an object instance (i.e. will print more than once if encountered in the same function name in different objects).

The Errors service provides methods to print both warnings and errors – essentially the same thing, but warning messages are prepended with ”WARNING:”, whereas error messages are prepended with ”ERROR:”.

Errors and warnings are printed by using the macros defined in ErrorsMacros.h. They are:

Error macros

SHOW_ERROR(error_number) Prints ”ERROR: ” followed by the error message associated with *error_number* (from the error catalog).

SHOW_ERROR(error_number, error_string) Prints ”ERROR: ” followed by the error message associated with *error_number* (from the error catalog), and appends ”error_string”.

SHOW_ERROR_IF(cond, error_number) If *cond* is true, prints ”ERROR: ” followed by the error message associated with *error_number* (from the error catalog).

SHOW_ERROR_IF(cond, error_number, error_string) If *cond* is TRUE, prints ”ERROR: ” followed by the error message associated with *error_number* (from the error catalog), and appends *error_string*.

Warning macros

SHOW_WARN(error_number) Prints "WARNING: " followed by the error message associated with *error_number* (from the error catalog).

SHOW_WARN(error_number, error_string) Prints "WARNING: " followed by the error message associated with *error_number* (from the error catalog), and appends *error_string*.

SHOW_WARN_IF(cond, error_number) If *cond* is TRUE, prints "WARNING: " followed by the error message associated with *error_number* (from the error catalog)

SHOW_WARN_IF(cond, error_number, error_string) If *cond* is TRUE, prints "WARNING: " followed by the error message associated with *error_number* (from the error catalog), and appends *error_string*

Error and warning message always contain:

The object id of the calling object.
The object type of the calling object.
The stellar type of the calling object (will be "NONE" if the calling object is not a star-type object).
The function name of the calling function.

Any object that uses the Errors service (i.e. the SHOW_* macros) must expose the following functions:

```
OBJECT_ID ObjectId() const { return m_ObjectId; }
OBJECT_TYPE ObjectType() const { return m_ObjectType; }
STELLAR_TYPE StellarType() const { return m_StellarType; }
```

These functions are called by the **SHOW_* macros**. If any of the functions are not applicable to the object, then they must return "*::NONE (all objects should implement ObjectId() correctly).

The filename to which error records are written when **Start()** parameter *errorsToLogfile* is true is declared in constants.h – see the LOGFILE enum class and associated descriptor map LOGFILE_DESCRIPTOR. Currently the name is "Error_Log".

Floating-Point Comparisons

Floating-point comparisons are inherently problematic. Testing floating-point numbers for equality, or even inequality, is fraught with problems due to the internal representation of floating-point numbers: floating-point numbers are stored with a fixed number of binary digits, which limits their precision and accuracy. The problems with floating-point comparisons are even more evident if one or both of the numbers being compared are the results of (perhaps several) floating-point operations (rather than comparing constants).

To avoid the problems associated with floating-point comparisons it is (almost always) better to do any such comparisons with a tolerance rather than an absolute comparison. To this end, a floating-point comparison function has been provided, and (almost all of) the floating-point comparisons in the code have been changed to use that function. The function uses both an absolute tolerance and a relative tolerance, which are both declared in constants.h. Whether the function uses a tolerance or not can be changed by #define-ing or #undef-ing the COMPARE_WITH_TOLERANCE flag in constants.h (so the change is a compile-time change, not run-time).

The compare function is defined in utils.h and is implemented as follows:

```
static int Compare(const double p_X, const double p_Y) {
#ifdef COMPARE_WITH_TOLERANCE
    return (fabs(p_X – p_Y) ≤ max( FLOAT_TOLERANCE_ABSOLUTE,
                                   FLOAT_TOLERANCE_RELATIVE *
                                   max( fabs(p_X),
                                        fabs(p_Y)))) ? 0 : (p_X < p_Y ? –1 : 1);
#else
    return (p_X == p_Y) ? 0 : (p_X < p_Y ? –1 : 1);
#endif
```

If COMPARE_WITH_TOLERANCE is defined, p_X and p_Y are compared with tolerance values, whereas if COMPARE_WITH_TOLERANCE is not defined the comparison is an absolute comparison.

The function returns an integer indicating the result of the comparison:

- –1 indicates that p_X is considered to be less than p_Y
- 0 indicates p_X and p_Y are considered to be equal
- +1 indicates that p_X is considered to be greater than p_Y

The comparison is done using both an absolute tolerance and a relative tolerance. The tolerances can be defined to be the same number, or different numbers. If the relative tolerance is defined as 0.0, the comparison is done using the absolute tolerance only, and if the absolute tolerance is defined as 0.0 the comparison is done with the relative tolerance only.

Absolute tolerances are generally more effective when the numbers being compared are small – so using an absolute tolerance of (say) 0.0000005 is generally effective when comparing single-digit numbers (or so), but is less effective when comparing numbers in the thousands or millions. For comparisons of larger numbers a relative tolerance is generally more effective (the actual tolerance is wider because the relative tolerance is multiplied by the larger absolute value of the numbers being compared).

There is a little overhead in the comparisons even when the tolerance comparison is disabled, but it shouldn't be prohibitive.

Constants File – constants.h

As well as plain constant values, many distribution and prescription identifiers are declared in constants.h. These are mostly declared as enum classes, with each enum class having a corresponding map of labels. The benefit is that the values of a particular (e.g.) prescription are limited to the values declared in the enum class, rather than any integer value, so the compiler will complain if an incorrect value is inadvertently used to reference that prescription.

For example, the Common Envelope Accretion Prescriptions are declared in constants.h thus:

```
enum class CE_ACCRETION_PRESCRIPTION: int {
    ZERO, CONSTANT, UNIFORM, MACLEOD
};

const std::unordered_map<CE_ACCRETION_PRESCRIPTION, std::string>
CE_ACCRETION_PRESCRIPTION_LABEL = {
    { CE_ACCRETION_PRESCRIPTION::ZERO,         "ZERO" },
    { CE_ACCRETION_PRESCRIPTION::CONSTANT, "CONSTANT" },
    { CE_ACCRETION_PRESCRIPTION::UNIFORM,   "UNIFORM" },
    { CE_ACCRETION_PRESCRIPTION::MACLEOD,  "MACLEOD" },
};
```

Note that the values allowed for variables of type CE_ACCRETION_PRESCRIPTION are limited to ZERO, CONSTANT, UNIFORM, and MACLEOD – anything else will cause a compilation error.

The unordered map CE_ACCRETION_PRESCRIPTION_LABEL declares a string label for each CE_ACCRETION_PRESCRIPTION, and is indexed by CE_ACCRETION_PRESCRIPTION. The strings declared in CE_ACCRETION_PRESCRIPTION_LABEL are used by the Options service to match user input to the required CE_ACCRETION_PRESCRIPTION. These strings can also be used if an English description of the value of a variable is required: instead of just printing an integer value that maps to a CE_ACCRETION_PRESCRIPTION, the string label associated with the prescription can be printed.

Stellar types are also declared in constants.h via an enum class and associate label map. This allows stellar types to be referenced using symbolic names rather than an ordinal number. The stellar types enum class is STELLAR_TYPE, and is declared as:

```
enum class STELLAR_TYPE: int {
    MS_LTE_07,
    MS_GT_07,
    HERTZSPRUNG_GAP,
    FIRST_GIANT_BRANCH,
    CORE_HELIUM_BURNING,
    EARLY_ASYMPTOTIC_GIANT_BRANCH,
    THERMALLY_PULSING_ASYMPTOTIC_GIANT_BRANCH,
    NAKED_HELIUM_STAR_MS,
    NAKED_HELIUM_STAR_HERTZSPRUNG_GAP,
    NAKED_HELIUM_STAR_GIANT_BRANCH,
    HELIUM_WHITE_DWARF,
    CARBON_OXYGEN_WHITE_DWARF,
    OXYGEN_NEON_WHITE_DWARF,
    NEUTRON_STAR,
```

```
        BLACK_HOLE,
        MASSLESS_REMNANT,
        CHEMICALLY_HOMOGENEOUS,
        STAR,
        BINARY_STAR,
        NONE
    };
```

Ordinal numbers can still be used to reference the stellar types, and because of the order of definition in the enum class the ordinal numbers match those given in Hurley et al. (2000).

The label map STELLAR_TYPE_LABEL can be used to print text descriptions of the stellar types, and is declared as:

```
const std::unordered_map<STELLAR_TYPE, std::string> STELLAR_TYPE_LABEL = {
  { STELLAR_TYPE::MS_LTE_07,                                      ”Main_Sequence_<=_0.7” },
  { STELLAR_TYPE::MS_GT_07,                                       ”Main_Sequence_>_0.7” },
  { STELLAR_TYPE::HERTZSPRUNG_GAP,                                ”Hertzsprung_Gap” },
  { STELLAR_TYPE::FIRST_GIANT_BRANCH,                             ”First_Giant_Branch” },
  { STELLAR_TYPE::CORE_HELIUM_BURNING,                            ”Core_Helium_Burning” },
  { STELLAR_TYPE::EARLY_ASYMPTOTIC_GIANT_BRANCH,                  ”Early_Asymptotic_Giant_Branch” },
  { STELLAR_TYPE::THERMALLY_PULSING_ASYMPTOTIC_GIANT_BRANCH,      ”Thermally_Pulsing_Asymptotic_Giant_Branch” },
  { STELLAR_TYPE::NAKED_HELIUM_STAR_MS,                           ”Naked_Helium_Star_MS” },
  { STELLAR_TYPE::NAKED_HELIUM_STAR_HERTZSPRUNG_GAP,              ”Naked_Helium_Star_Hertzsprung_Gap” },
  { STELLAR_TYPE::NAKED_HELIUM_STAR_GIANT_BRANCH,                 ”Naked_Helium_Star_Giant_Branch” },
  { STELLAR_TYPE::HELIUM_WHITE_DWARF,                             ”Helium_White_Dwarf” },
  { STELLAR_TYPE::CARBON_OXYGEN_WHITE_DWARF,                      ”Carbon-Oxygen_White_Dwarf” },
  { STELLAR_TYPE::OXYGEN_NEON_WHITE_DWARF,                        ”Oxygen-Neon_White_Dwarf” },
  { STELLAR_TYPE::NEUTRON_STAR,                                   ”Neutron_Star” },
  { STELLAR_TYPE::BLACK_HOLE,                                     ”Black_Hole” },
  { STELLAR_TYPE::MASSLESS_REMNANT,                               ”Massless_Remnant” },
  { STELLAR_TYPE::CHEMICALLY_HOMOGENEOUS,                         ”Chemically_Homogeneous” },
  { STELLAR_TYPE::STAR,                                           ”Star” },
  { STELLAR_TYPE::BINARY_STAR,                                    ”Binary_Star” },
  { STELLAR_TYPE::NONE,                                           ”Not_a_Star!” },
};
```

Programming Style and Conventions

Everyone has their own preferences and style, and the nature of a project such as COMPAS will reflect that. However, there is a need to suggest some guidelines for programming style, naming conventions etc. Following is a description of some of the elements of programming style and naming conventions used to develop COMPAS v2. These may evolve over time.

Object-Oriented Programming

COMPAS is written in C++, an object-oriented programming (OOP) language, and OOP concepts and conventions should apply throughout the code. There are many texts and web pages devoted to understanding C++ and OOP – following is a brief description of the key OOP concepts:

Abstraction

For any entity, product, or service, the goal of abstraction is to handle the complexity of the implementation by hiding details that don't need to be known in order to use, or consume, the entity, product, or service. In the OOP paradigm, hiding details in this way enables the consumer to implement more complex logic on top of the provided abstraction without needing to understand the hidden implementation details and complexity. (There is no suggestion that consumers shouldn't understand the implementation details, but they shouldn't need to in order to consume the entity, product, or service).

Abstraction in C++ is achieved via the use of *objects* – an object is an instance of a *class*, and typically corresponds to a real-world object or entity (in COMPAS, usually a star or binary star). An object maintains the state of an object (via class member variables), and provides all necessary means of changing the state of the object (by exposing public class member functions (methods)). A class may expose public functions to allow consumers to determine the value of class member variables ("getters"), and to set the value of class member variables ("setters").

Encapsulation

Encapsulation binds together the data and functions that manipulate the data in an attempt to keep both safe from outside interference and accidental misuse. An encapsulation paradigm that does not allow calling code to access internal object data and permits access through functions only is a strong form of abstraction. C++ allows developers to enforce access restrictions explicitly by defining class member variables and functions as *private*, *protected*, or *public*. These keywords are used throughout COMPAS to enforce encapsulation.

There are very few circumstances in which a consumer should change the value of a class member variable directly (via the use of a setter function) – almost always consumers should present new situational information to an object (via a public member function), and allow the object to respond to the new information. For example, in COMPAS, there should be almost no reason for a consumer of a star object to directly change (say) the radius of the star – the consumer should inform the star object of new circumstances or events, and allow the star object to respond to those events (perhaps changing the value of the radius of the star). Changing a single class member variable directly introduces the possibility that related class member variables (e.g. other attributes of stars) will not be changed accordingly. Moreover, developers changing the code in the future should, in almost all cases, expect that the state of an object is maintained consistently by the object, and that there should be no unexpected side-effects caused by calling non class-member functions.

In short, changing the state of an object outside the object is potentially unsafe and should be avoided where possible.

Inheritance

Inheritance allows classes to be arranged in a hierarchy that represents *is-a-type-of* relationships. All *non-private* class member variables and functions of the parent (base) class are available to the child (derived) class (and, therefore, child classes of the child class). This allows easy re-use of the same procedures and data definitions, in addition to describing real-world relationships in an intuitive way. C++ allows multiple inheritance – a class may inherit from multiple parent classes.

Derived classes can define additional class member variables (using the *private*, *protected*, and *public* access restrictions), which will be available to any descendent classes (subject to inheritance rules), but will only be available to ancestor classes via the normal access methods (getters and setters).

Polymorphism

Polymorphism means *having many forms*. In OOP, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

Following the discussion above regarding inheritance, in the OOP paradigm, and C++ specifically, derived classes can override methods defined by ancestor classes, allowing a derived class to implement functions specific to its circumstances. This means that a call to a class member function will cause a different function to be executed depending on the type of object that invokes the function. Descendent classes of a class that has overridden a base class member function inherit the overridden function (but can override it themselves).

COMPAS makes heavy use of inheritance and polymorphism, especially for the implementation of the different stellar types.

Programming Style

The goal of coding to a suggested style is readability and maintainability – if many developers implement code in COMPAS with their own coding style, readability and maintainability will be more difficult than if a consistent style is used throughout the code. Strict adherence isn't really necessary, but it will make it easier on all COMPAS developers if the coding style is consistent throughout.

Comments

An old, but good, rule-of-thumb is that any file that contains computer code should be about one-third code, one-third comments, and one-third white space. Adhering to this rule-of-thumb just makes the code a bit easier on the eye, and provides some description (at least of the intention) of the implementation.

Braces

The placement of braces in C++ code (actually, any code that uses braces to enclose scope) is a contentious issue, with many developers having long-held, often dogmatic preferences. COMPAS (so far) uses the K&R

style (”the one true brace style”) - the style used in the original Unix kernel and Kernighan and Ritchie’s book *The C Programming Language*.

The K&R style puts the opening brace on the same line as the control statement:

```
    while (x == y) {
        something();
        somethingelse();
}
```

Note also the space between the keyword *while* and the opening parenthesis, surrounding the == operator, and between the closing parenthesis and the opening brace. Spaces here helps with code readability. Surrounding all arithmetic operators with spaces is preferred.

Indentation

There is ongoing debate in the programming community as to whether indentation should be achieved using spaces or tabs (strange, but true…). The use of spaces is more common. COMPAS (so far) has a mix of both – whatever is convenient (pragmatism is your friend...). Unfortunately a mix of spaces and tabs doesn’t work well with some editors - we should settle on one method and try to stick to it.

COMPAS (mostly) uses an indentation size of 4 spaces - again we should settle on a soze and stick to it.

Function Parameters

In most cases, function parameters should be input only – meaning that the values of function parameters should not be changed by the function. Anything that needs to be changed and returned to the caller should be returned as a functional return. There are a few exceptions to this in COMPAS – all were done for performance reasons, and are documented in the code.

To avoid unexpected side-effects, developers should expect (in most cases) that any variables they pass to a function will remain unchanged – all changes should be returned as a functional return.

Performance & Optimisation

In general, COMPAS developers should code for performance – within reason. Bear in mind that many functions will be called many, many thousands of times (in some cases, millions) in one execution of the program.

- Avoid calculating values inside loops that could be calculated once outside the loop.
- Try to use constants where possible.
- Use multiplication in preference to functions such as *pow()* and *sqrt()* (note that *pow()* is very expensive computationally; *sqrt()* is expensive, but much less expensive than *pow()*).
- Don’t optimise to the point that readability and maintainability is compromised. Bear in mind that most compilers are good at optimising, and are very forgiving of less-than-optimally-written code (though they are not miracle workers...).

Naming Conventions

COMPAS (so far) uses the following naming conventions:

- All variable names should be in camelCase – don't use underscore_to_separate_words.
- Function names should be in camelCase, beginning with an uppercase letter. Function names should be descriptive.
- Class member variable names are prefixed with "m_", and the character immediately following the prefix should be uppercase (in most cases – sometimes, for well-known names or words that are always written in lowercase, lowercase might be used).
- Local variable names are just camelCase, beginning with a lowercase letter (again, with the caveat that sometimes, for well-known names or words that are always written in uppercase, uppercase might be used).
- Function parameter names are prefixed with "p_", and the character immediately following the prefix should be uppercase (again, with the caveat that sometimes, for well-known names or words that are always written in lowercase, lowercase might be used).

Compilation & Requirements

Please refer to the COMPAS Getting Started guide.

References

Barrett, J. W., Gaebel, S. M., Neijssel, C. J., et al. 2018, MNRAS, 477, 4685

Belczynski, K., Kalogera, V., Rasio, F. A., et al. 2008, ApJS, 174, 223

Broekgaarden, F. S., Justham, S., de Mink, S. E., et al. 2019, MNRAS, 490, 5228

Chattopadhyay, D., Stevenson, S., Hurley, J. R., Rossi, L. J., & Flynn, C. 2020, MNRAS, 494, 1587

Claeys, J. S. W., Pols, O. R., Izzard, R. G., Vink, J., & Verbunt, F. W. M. 2014, A & A, 563, A83

Dewi, J., & Tauris, T. 2000, A & A, 360

Fryer, C. L., Belczynski, K., Wiktorowicz, G., et al. 2012, Apj, 749, 91

Hurley, J. R., Pols, O. R., & Tout, C. A. 2000, MNRAS, 315, 543

Kruckow, M. U., Tauris, T. M., Langer, N., et al. 2016, A & A, 596, A58

Loveridge, A. J., van der Sluys, M. V., & Kalogera, V. 2011, Apj, 743, 49

Mennekens, N., & Vanbeveren, D. 2014, A & A, 564, A134

Neijssel, C. J., Vigna-Gómez, A., Stevenson, S., et al. 2019, MNRAS, 490, 3740

Soberman, G. E., Phinney, E. S., & van den Heuvel, E. P. J. 1997, A & A, 327, 620

Stevenson, S., Sampson, M., Powell, J., et al. 2019, Apj, 882, 121

Stevenson, S., Vigna-Gómez, A., Mandel, I., et al. 2017, Nat. Commun., 8, 14906

Vigna-Gómez, A., Neijssel, C. J., Stevenson, S., et al. 2018, MNRAS, 481, 4009–4029

Webbink, R. F. 1984, Apj, 277, 355

Xu, X.-J., & Li, X.-D. 2010, Apj, 716, 114

Program Options

Following is the list of program options that can be specified at the command line when running COMPAS.

--help [-h]
Prints COMPAS help.

--version [-v]
Prints COMPAS version string.

--add-options-to-sysparms
Add columns for program options to SSE_System_Parameters/BSE_System_Parameters file (mode dependent).
Options: { ALWAYS, GRID, NEVER }
Default = GRID

ALWAYS indicates that the program options should be added to the sysparms file
GRID indicates that the program options should be added to the sysparms file only if a GRID file is specified, or RANGEs or SETs are specified for options
NEVER indicates that the program options should not be added to the sysparms file

--allow-rlof-at-birth
Allow binaries that have one or both stars in RLOF at birth to evolve as over-contact systems.
Default = FALSE

--allow-touching-at-birth
Allow binaries that are touching at birth to be included in the sampling.
Default = FALSE

--angularMomentumConservationDuringCircularisation
Conserve angular momentum when binary is circularised when entering a Mass Transfer episode.
Default = FALSE

--black-hole-kicks
Black hole kicks relative to NS kicks.
Options: { FULL, REDUCED, ZERO, FALLBACK }
Default = FALLBACK

--case-bb-stability-prescription
Prescription for the stability of case BB/BC mass transfer.
Options: { ALWAYS_STABLE, ALWAYS_STABLE_ONTO_NSBH, TREAT_AS_OTHER_MT, NEVER_STABLE }
Default = ALWAYS_STABLE

--check-photon-tiring-limit
Check the photon tiring limit is not exceeded during mass loss.
Default = FALSE

--chemically-homogeneous-evolution
Chemically Homogeneous Evolution mode.
Options: { NONE, OPTIMISTIC, PESSIMISTIC }
Default = NONE

--circulariseBinaryDuringMassTransfer
Circularise binary when it enters a Mass Transfer episode.
Default = FALSE

--common-envelope-allow-main-sequence-survive
Allow main sequence donors to survive common envelope evolution.
Default = FALSE

--common-envelope-alpha
Common Envelope efficiency alpha.
Default = 1.0

--common-envelope-alpha-thermal
Thermal energy contribution to the total envelope binding energy.
Defined such that $\lambda = \alpha_{th} \times \lambda_b + (1.0 - \alpha_{th}) \times \lambda_g$.
Default = 1.0

--common-envelope-lambda
Common Envelope lambda.
Default = 0.1

--common-envelope-lambda-multiplier
Multiplication constant to be applied to the common envelope lambda parameter.
Default = 1.0

--common-envelope-lambda-prescription
CE lambda prescription.
Options: { LAMBDA_FIXED, LAMBDA_LOVERIDGE, LAMBDA_NANJING, LAMBDA_KRUCKOW, LAMBDA_DEWI }
Default = LAMBDA_NANJING

--common-envelope-mass-accretion-constant
Value of mass accreted by NS/BH during common envelope evolution if assuming all NS/BH accrete same amount of mass.
Used when *--common-envelope-mass-accretion-prescription* = *CONSTANT*, ignored otherwise.
Default = 0.0

--common-envelope-mass-accretion-max
Maximum amount of mass accreted by NS/BHs during common envelope evolution ($M_\odot$).
Default = 0.1

--common-envelope-mass-accretion-min
Minimum amount of mass accreted by NS/BHs during common envelope evolution ($M_\odot$).
Default = 0.04

--common-envelope-mass-accretion-prescription
Assumption about whether NS/BHs can accrete mass during common envelope evolution.
Options: { ZERO, CONSTANT, UNIFORM, MACLEOD }
Default = ZERO

--common-envelope-recombination-energy-density
Recombination energy density (erg g^{-1}).
Default = 1.5×10^{13}

--common-envelope-slope-Kruckow
Common Envelope slope for Kruckow lambda.
Default = -0.8

--cool-wind-mass-loss-multiplier
Multiplicative constant for wind mass loss of cool stars, i.e. those with temperatures below the VINK_MASS_LOSS_MINIMUM_TEMP (default 12500K).
Only applicable when mass-loss-prescription is set to VINK.
Default = 1.0

--debug-classes
Debug classes enabled.
Default = ’’ (None)

--debug-level
Determines which print statements are displayed for debugging.
Default = 0

--debug-to-file
Write debug statements to file.
Default = FALSE

--detailedOutput
Print BSE detailed information to file.
Default = FALSE

--eccentricity [-e]
Initial eccentricity for a binary star when evolving in BSE mode.
Default = 0.0

--eccentricity-distribution
Initial eccentricity distribution.
Options: { ZERO, FLAT, GELLER+2013, THERMAL, DUQUENNOYMAYOR1991, SANA2012 }
Default = ZERO

--eccentricity-max
Maximum eccentricity to generate.
Default = 1.0

--eccentricity-min
Minimum eccentricity to generate.
Default = 0.0

--eddington-accretion-factor
Multiplication factor for Eddington accretion for NS & BH (i.e. >1 is super-eddington and 0 is no accretion).
Default = 1.0

--envelope-state-prescription
Prescription for determining whether the envelope of the star is convective or radiative.
Options: { LEGACY, HURLEY, FIXED_TEMPERATURE }
Default = LEGACY

--errors-to-file
Write error messages to file.
Default = FALSE

--evolve-pulsars
Evolve pulsar properties of Neutron Stars.
Default = FALSE

--evolve-unbound-systems
Continue evolving stars even if the binary is disrupted.
Default = FALSE

--fix-dimensionless-kick-magnitude
Fix dimensionless kick magnitude to this value.
Default = n/a (not used if option not present)

--fryer-supernova-engine
Supernova engine type if using the fallback prescription from Fryer et al. (2012).
Options: { DELAYED, RAPID }
Default = DELAYED

--grid
Grid filename.
Default = ’’ (None)

--hdf5-buffer-size
The HDF5 IO buffer size for writing to HDF5 logfiles (number of HDF5 chunks).
Default = 1

--hdf5-chunk-size
The HDF5 dataset chunk size to be used when creating HDF5 logfiles (number of logfile entries).
Default = 100000

--initial-mass
Initial mass for a single star when evolving in SSE mode ($M_\odot$).
Default = Sampled from IMF

--initial-mass-1
Initial mass for the primary star when evolving in BSE mode ($M_\odot$).
Default = Sampled from IMF

--initial-mass-2
Initial mass for the secondary star when evolving in BSE mode ($M_\odot$).
Default = Sampled from IMF

--initial-mass-function [-i]
Initial mass function.
Options: { SALPETER, POWERLAW, UNIFORM, KROUPA }
Default = KROUPA

--initial-mass-max
Maximum mass to generate using given IMF ($M_\odot$).
Default = 100.0

--initial-mass-min
Minimum mass to generate using given IMF ($M_\odot$).
Default = 8.0

--initial-mass-power
Single power law power to generate primary mass using POWERLAW IMF.
Default = –2.3

--kick-direction
Natal kick direction distribution.
Options: { ISOTROPIC, INPLANE, PERPENDICULAR, POWERLAW, WEDGE, POLES }
Default = ISOTROPIC

--kick-direction-power
Power for power law kick direction distribution, where 0.0 = isotropic, +ve = polar, -ve = in plane.
Default = 0.0 (isotropic)

--kick-scaling-factor
Arbitrary factor used to scale kicks.
Default = 1.0

--kick-magnitude
Value to be used as the (drawn) kick magnitude for a single star when evolving in SSE mode, should the star undergo a supernova event (km s^{-1}).
If a value for option *--kick-magnitude-random* is specified, it will be used in preference to *--kick-magnitude*.
Default = 0.0

--kick-magnitude-1
Value to be used as the (drawn) kick magnitude for the primary star of a binary system when evolving in BSE mode, should the star undergo a supernova event (km s^{-1}).
If a value for option *--kick-magnitude-random-1* is specified, it will be used in preference to *--kick-magnitude-1*.
Default = 0.0

--kick-magnitude-2
Value to be used as the (drawn) kick magnitude for the secondary star of a binary system when evolving in BSE mode, should the star undergo a supernova event (km s^{-1}).
If a value for option *--kick-magnitude-random-2* is specified, it will be used in preference to *--kick-magnitude-2*.
Default = 0.0

--kick-magnitude-distribution
Natal kick magnitude distribution.
Options: { ZERO, FIXED, FLAT, MAXWELLIAN, BRAYELDRIDGE, MULLER2016, MULLER2016MAXWELLIAN, MULLERMANDEL }
Default = MAXWELLIAN

--kick-magnitude-max
Maximum drawn kick magnitude (km s^{-1}).
Must be > 0 if using *--kick-magnitude-distribution* = *FLAT*.
Default = -(1.0)

--kick-magnitude-random
Value to be used to draw the kick magnitude for a single star when evolving in SSE mode, should the star undergo a supernova event.
Must be a floating-point number in the range [0.0, 1.0).
The specified value for this option will be used in preference to any specified value for *--kick-magnitude*.
Default = Random number drawn uniformly from [0.0, 1.0)

--kick-magnitude-random-1
Value to be used to draw the kick magnitude for the primary star of a binary system when evolving in BSE mode, should the star undergo a supernova event.
Must be a floating-point number in the range [0.0, 1.0).
The specified value for this option will be used in preference to any specified value for *--kick-magnitude-1*.
Default = Random number drawn uniformly from [0.0, 1.0)

--kick-magnitude-random-2
Value to be used to draw the kick magnitude for the secondary star of a binary system when evolving in BSE mode, should the star undergo a supernova event.
Must be a floating-point number in the range [0.0, 1.0).
The specified value for this option will be used in preference to any specified value for *--kick-magnitude-2*.
Default = Random number drawn uniformly from [0.0, 1.0)

--kick-magnitude-sigma-CCSN-BH
Sigma for chosen kick magnitude distribution for black holes (km s^{-1}).
Default = 250.0

--kick-magnitude-sigma-CCSN-NS
Sigma for chosen kick magnitude distribution for neutron stars (km s^{-1}).
Default = 250.0

--kick-magnitude-sigma-ECSN
Sigma for chosen kick magnitude distribution for ECSN (km s^{-1}).
Default = 30.0

--kick-magnitude-sigma-USSN
Sigma for chosen kick magnitude distribution for USSN (km s^{-1}).
Default = 30.0

--kick-mean-anomaly-1
The mean anomaly at the instant of the supernova for the primary star of a binary system when evolving in BSE mode, should it undergo a supernova event.
Must be a floating-point number in the range [0.0, 2π).
Default = Random number drawn uniformly from [0.0, 2π)

--kick-mean-anomaly-2
The mean anomaly at the instant of the supernova for the secondary star of a binary system when evolving in BSE mode, should it undergo a supernova event.
Must be a floating-point number in the range [0.0, 2π).
Default = Random number drawn uniformly from [0.0, 2π)

--kick-phi-1
The angle between 'x' and 'y', both in the orbital plane of the supernova vector, for the for the primary star of a binary system when evolving in BSE mode, should it undergo a supernova event (radians).
Default = Random number drawn uniformly from [0.0, 2π)

--kick-phi-2
The angle between 'x' and 'y', both in the orbital plane of the supernova vector, for the for the secondary star of a binary system when evolving in BSE mode, should it undergo a supernova event (radians).
Default = Random number drawn uniformly from [0.0, 2π)

--kick-theta-1
The angle between the orbital plane and the 'z' axis of the supernova vector for the for the primary star of a binary system when evolving in BSE mode, should it undergo a supernova event (radians).
Default = Random number drawn uniformly from [0.0, 2π)

--log-classes
Logging classes enabled.
Default = '' (None)

--logfile-common-envelopes
Filename for BSE Common Envelopes logfile.
Default = 'BSE_Common_Envelopes'

--logfile-definitions
Filename for logfile record definitions file.
Default = '' (None)

--logfile-detailed-output
Filename for the Detailed Output logfile.
Default = 'SSE_Detailed_Output' for SSE mode; 'BSE_Detailed_Output' for BSE mode

--logfile-double-compact-objects
Filename for the Double Compact Objects logfile (BSE mode).
Default = 'BSE_Double_Compact_Objects'

--logfile-name-prefix
Prefix for logfile names.
Default = '' (None)

--logfile-pulsar-evolution
Filename for the Pulsar Evolution logfile (BSE mode).
Default = 'BSE_Pulsar_Evolution'

--logfile-rlof-parameters
Filename for the RLOF Printing logfile (BSE mode).
Default = 'BSE_RLOF'

--logfile-supernovae
Filename for the Supernovae logfile.
Default = 'SSE_Supernovae' for SSE mode; 'BSE_Supernovae' for BSE mode

--logfile-switch-log
Filename for the Switch Log logfile.
Default = 'SSE_Switch_Log' for SSE mode; 'BSE_Switch_Log' for BSE mode

--logfile-system-parameters
Filename for the System Parameters logfile (BSE mode).
Default = 'BSE_System_Parameters'

--logfile-type
The type of logfile to be produced by COMPAS.
Default = 'HDF5'

--log-level
Determines which print statements are included in the logfile.
Default = 0

--luminous-blue-variable-prescription
Luminous blue variable mass loss prescription.
Options: { NONE, HURLEY, HURLEY_ADD, BELCZYNSKI }
Default = HURLEY_ADD

--luminous-blue-variable-multiplier
Multiplicative constant for LBV mass loss. (Use 10 for Mennekens & Vanbeveren (2014)) Note that wind mass loss will also be multiplied by the overall-wind-mass-loss-multiplier.
Default = 1.5

--mass-loss-prescription
Mass loss prescription.
Options: { NONE, HURLEY, VINK }
Default = VINK

--mass-ratio [-q]
Mass ratio $\frac{m2}{m1}$ used to determine secondary mass if not specified via *--initial-mass-2*.
Default = Value is sampled if option not specified.

--mass-ratio-distribution
Initial mass ratio distribution for q= $\frac{m2}{m1}$.
Options: { FLAT, DUQUENNOYMAYOR1991, SANA2012 }
Default = FLAT

--mass-ratio-max
Maximum mass ratio $\frac{m2}{m1}$ to generate.
Default = 1.0

--mass-ratio-min
Minimum mass ratio $\frac{m2}{m1}$ to generate.
Default = 0.01

--massTransfer
Enable mass transfer.
Default = TRUE

--mass-transfer-accretion-efficiency-prescription
Mass transfer accretion efficiency prescription.
Options: { THERMAL, FIXED, CENTRIFUGAL }
Default = ISOTROPIC

--mass-transfer-angular-momentum-loss-prescription
Mass Transfer Angular Momentum Loss prescription.
Options: { JEANS, ISOTROPIC, CIRCUMBINARY, ARBITRARY }
Default = ISOTROPIC

--mass-transfer-fa
Mass Transfer fraction accreted.
Used when
--mass-transfer-accretion-efficiency-prescription = *FIXED_FRACTION*.
Default = 1.0 (fully conservative)

--mass-transfer-jloss
Specific angular momentum with which the non-accreted system leaves the system.
Used when *--mass-transfer-angular-momentum-loss-prescription* = *ARBITRARY*, ignored otherwise.
Default = 1.0

--mass-transfer-thermal-limit-accretor
Mass Transfer Thermal Accretion limit multiplier.
Options: { CFACTOR, ROCHELOBE }

--mass-transfer-thermal-limit-C
Mass Transfer Thermal rate factor for the accretor.
Default = 10.0

--mass-transfer-rejuvenation-prescription
Mass Transfer Rejuvenation prescription.
Options: { NONE, STARTRACK }
Default = NONE

--maximum-evolution-time
Maximum time to evolve binaries (Myr). Evolution of the binary will stop if this number is reached.
Default = 13700.0

--maximum-mass-donor-Nandez-Ivanova
Maximum donor mass allowed for the revised common envelope formalism of Nandez & Ivanova ($M_\odot$).
Default = 2.0

--maximum-neutron-star-mass
Maximum mass of a neutron star ($M_\odot$).
Default = 3.0

--maximum-number-timestep-iterations
Maximum number of timesteps to evolve binary. Evolution of the binary will stop if this number is reached.
Default = 99999

--mcbur1
Minimum core mass at base of AGB to avoid fully degnerate CO core formation ($M_\odot$).
e.g. 1.6 in Hurley et al. (2000) presciption; 1.83 in Fryer et al. (2012) and Belczynski et al. (2008) models.
Default = 1.6)

--metallicity [-z]
Metallicity.
The value specified for metallicity is applied to both stars for BSE mode.
Default = 0.02

--metallicity-distribution
Metallicity distribution.
Options: { ZSOLAR, LOGUNIFORM }
Default = ZSOLAR

--metallicity-max
Maximum metallicity to generate.
Default = 0.04

--metallicity-min
Minimum metallicity to generate.
Default = 0.0001

--minimum-secondary-mass
Minimum mass of secondary to generate ($M_\odot$).
Default = 0.00007 if *--initial-mass-2* specified; value of *--initial-mass-min* if *--initial-mass-2* not specified.

--mode
The mode of evolution.
Options: { SSE, BSE }
Default = BSE

--neutrino-mass-loss-bh-formation
Assumption about neutrino mass loss during BH formation.
Options: { FIXED_FRACTION, FIXED_MASS }
Default = FIXED_FRACTION

--neutrino-mass-loss-bh-formation-value
Amount of mass lost in neutrinos during BH formation (either as fraction or in solar masses, depending on the value of *--neutrino-mass-loss-bh-formation*).
Default = 0.1

--neutron-star-equation-of-state
Neutron star equation of state.
Options: { SSE, ARP3 }
Default = SSE

--number-of-systems [-n]
The number of systems to simulate.
Single stars for SSE mode; binary stars for BSE mode.
This option is ignored if either of the following is true:
- the user specified a grid file
- the user specified a range or set for any options - this implies a grid

In both cases the number of objects evolved will be the number specified by the grid.
Default = 10

--orbital-period
Initial orbital period for a binary star when evolving in BSE mode (days).
Used only if the semi-major axis is not specified via *--semi-major-axis*.
Default = Value is sampled if option not specified.

--orbital-period-max
Maximum period to generate (days).
Default = 1000.0

--orbital-period-min
Minimum period to generate (days).
Default = 1.1

--output-container [-c]
Container (directory) name for output files.
Default = 'COMPAS_Output'

--output-path [-o]
Path to which output is saved (i.e. directory in which the output container is created).
Default = Current working directory (CWD)

--overall-wind-mass-loss-multiplier
Multiplicative constant for overall wind mass loss.
Note that this multiplication factor is applied after the luminous-blue-variable-, the wolf-rayet- and the cool-wind-mass-loss-multiplier.
Default = 1.0

--pair-instability-supernovae
Enable pair instability supernovae (PISN).
Default = FALSE

--PISN-lower-limit
Minimum core mass for PISN ($M_\odot$).
Default = 60.0

--PISN-upper-limit
Maximum core mass for PISN ($M_\odot$).
Default = 135.0

--population-data-printing
Print details of population.
Default = FALSE

--PPI-lower-limit
Minimum core mass for PPI ($M_\odot$).
Default = 35.0

--PPI-upper-limit
Maximum core mass for PPI ($M_\odot$).
Default = 60.0

--print-bool-as-string
Print boolean properties as 'TRUE' or 'FALSE'.
Default = FALSE

--pulsar-birth-magnetic-field-distribution
Pulsar birth magnetic field distribution.
Options: { ZERO, FIXED, FLATINLOG, UNIFORM, LOGNORMAL }
Default = ZERO

--pulsar-birth-magnetic-field-distribution-max
Maximum ($\log_{10}$) pulsar birth magnetic field.
Default = 13.0

--pulsar-birth-magnetic-field-distribution-min
Minimum ($\log_{10}$) pulsar birth magnetic field.
Default = 11.0

--pulsar-birth-spin-period-distribution
Pulsar birth spin period distribution.
Options: { ZERO, FIXED, UNIFORM, NORMAL }
Default = ZERO

--pulsar-birth-spin-period-distribution-max
Maximum pulsar birth spin period (ms).
Default = 100.0

--pulsar-birth-spin-period-distribution-min
Minimum pulsar birth spin period (ms).
Default = 0.0

--pulsar-magnetic-field-decay-massscale
Mass scale on which magnetic field decays during accretion ($M_\odot$).
Default = 0.025

--pulsar-magnetic-field-decay-timescale
Timescale on which magnetic field decays (Myr).
Default = 1000.0

--pulsar-minimum-magnetic-field
$\log_{10}$ of the minimum pulsar magnetic field (Gauss).
Default = 8.0

--pulsational-pair-instability
Enable mass loss due to pulsational-pair-instability (PPI).
Default = FALSE

--pulsational-pair-instability-prescription
Pulsational pair instability prescription.
Options: { COMPAS, STARTRACK, MARCHANT, FARMER }
Default = COMPAS

--quiet
Suppress printing to stdout.
Default = FALSE

--random-seed
Value to use as the seed for the random number generator.
Default = 0

--remnant-mass-prescription
Remnant mass prescription.
Options: { HURLEY2000, BELCZYNSKI2002, FRYER2012, MULLER2016, MULLERMANDEL, SCHNEIDER2020, SCHNEIDER2020ALT }
Default = FRYER2012

--revised-energy-formalism-Nandez-Ivanova
Enable revised energy formalism of Nandez & Ivanova.
Default = FALSE

--RLOFprinting
Print RLOF events to logfile.
Default = FALSE

--rotational-frequency
Initial rotational frequency of the star for SSE (Hz).
Default = 0.0 (–rotational-velocity-distribution used if –rotational-frequency not specified)

--rotational-frequency-1
Initial rotational frequency of the primary star for BSE (Hz).
Default = 0.0 (–rotational-velocity-distribution used if –rotational-frequency-1 not specified)

--rotational-frequency-2
Initial rotational frequency of the secondary star for BSE (Hz).
Default = 0.0 (–rotational-velocity-distribution used if –rotational-frequency-2 not specified)

--rotational-velocity-distribution
Initial rotational velocity distribution.
Options: { ZERO, HURLEY, VLTFLAMES }
Default = ZERO

--semi-major-axis
Initial semi-major axis for a binary star when evolving in BSE mode (AU).
Default = 1000.0

--semi-major-axis-distribution [-a]
Initial semi-major axis distribution.
Options: { FLATINLOG, CUSTOM, DUQUENNOYMAYOR1991, SANA2012 }
Default = FLATINLOG

--semi-major-axis-max
Maximum semi-major axis to generate (AU).
Default = 1000.0

--semi-major-axis-min
Minimum semi-major axis to generate (AU).
Default = 0.1

--stellar-zeta-prescription
Prescription for stellar zeta.
Options: { STARTRACK, SOBERMAN, HURLEY, ARBITRARY }
Default = SOBERMAN

--switch-log
Enables printing of the Switch Log logfile
Default = FALSE

--timestep-multiplier
Multiplicative factor for timestep duration
Default = 1.0

--use-mass-loss
Enable mass loss.
Default = FALSE

--wolf-rayet-multiplier
Multiplicative constant for Wolf Rayet winds. Note that wind mass loss will also be multiplied by the overall-wind-mass-loss-multiplier.
Default = 1.0

--zeta-adiabatic-arbitrary
Value of logarithmic derivative of radius with respect to mass, ζ adiabatic.
Default = 1.0×10^4

--zeta-main-sequence
Value of logarithmic derivative of radius with respect to mass, ζ on the main sequence.
Default = 2.0

--zeta-radiative-giant-star
Value of logarithmic derivative of radius with respect to mass, ζ for radiative-envelope giant-like stars (including Hertzsprung Gap (HG) stars).
Default = 6.5

Log File Record Specification: Stellar Properties

As described in Section **Extended Logging**, when specifying known properties in a log file record specification record, the property name must be prefixed with the property type.

The current list of valid stellar property types available for use is:

- STAR_PROPERTY for SSE
- STAR_1_PROPERTY for the primary star of a binary for BSE
- STAR_2_PROPERTY for the secondary star of a binary for BSE
- SUPERNOVA_PROPERTY for the exploding star in a supernova event for BSE
- COMPANION_PROPERTY for the companion star in a supernova event for BSE

For example, to specify the property TEMPERATURE for an individual star being evolved for SSE, use:

STAR_PROPERTY::TEMPERATURE

To specify the property TEMPERATURE for the primary star in a binary star being evolved for BSE, use:

STAR_1_PROPERTY::TEMPERATURE

To specify the property TEMPERATURE for the supernova star in a binary star being evolved for BSE, use:

SUPERNOVA_PROPERTY::TEMPERATURE

Following is the list of stellar properties available for inclusion in log file record specifiers.

Stellar Properties

AGE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Age
Description:	Effective age (changes with mass loss/gain) (Myr)
Header Strings:	Age, Age(1), Age(2), Age(SN), Age(CP)

ANGULAR_MOMENTUM

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_AngularMomentum
Description:	Angular momentum ($M_\odot$ AU^2 yr^{-1})
Header Strings:	Ang_Momentum, Ang_Momentum(1), Ang_Momentum(2), Ang_Momentum(SN), Ang_Momentum(CP)

BINDING_ENERGY_AT_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.bindingEnergy
Description:	Absolute value of the envelope binding energy at the onset of unstable RLOF (erg). Used for calculating post-CE separation. *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Binding_Energy@CE(1), Binding_Energy@CE(2), Binding_Energy@CE(SN), Binding_Energy@CE(CP)

BINDING_ENERGY_FIXED

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_BindingEnergies.fixed
Description:	Absolute value of the envelope binding energy calculated using a fixed lambda parameter (erg). Calculated using lambda = m_Lambdas.fixed.
Header Strings:	BE_Fixed, BE_Fixed(1), BE_Fixed(2), BE_Fixed(SN), BE_Fixed(CP)

BINDING_ENERGY_KRUCKOW

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_BindingEnergies.kruckow
Description:	Absolute value of the envelope binding energy calculated using the fit by Vigna-Gómez et al. (2018) to Kruckow et al. (2016) (erg). Calculated using alpha = OPTIONS→CommonEnvelopeSlopeKruckow().
Header Strings:	BE_Kruckow, BE_Kruckow(1), BE_Kruckow(2), BE_Kruckow(SN), BE_Kruckow(CP)

BINDING_ENERGY_LOVERIDGE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_BindingEnergies.loveridge
Description:	Absolute value of the envelope binding energy calculated as per Loveridge et al. (2011) (erg). Calculated using lambda = m_Lambdas.loveridge.
Header Strings:	BE_Loveridge, BE_Loveridge(1), BE_Loveridge(2), BE_Loveridge(SN), BE_Loveridge(CP)

BINDING_ENERGY_LOVERIDGE_WINDS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_BindingEnergies.loveridgeWinds
Description:	Absolute value of the envelope binding energy calculated as per Webbink (1984) & Loveridge et al. (2011) including winds (erg). Calculated using lambda = m_Lambdas.loveridgeWinds.
Header Strings:	BE_Loveridge_Winds, BE_Loveridge_Winds(1), BE_Loveridge_Winds(2), BE_Loveridge_Winds(SN), BE_Loveridge_Winds(CP)

BINDING_ENERGY_NANJING

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_BindingEnergies.nanjing
Description:	Absolute value of the envelope binding energy calculated as per Xu & Li (2010) (erg). Calculated using lambda = m_Lambdas.nanjing.
Header Strings:	BE_Nanjing, BE_Nanjing(1), BE_Nanjing(2), BE_Nanjing(SN), BE_Nanjing(CP)

BINDING_ENERGY_POST_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.postCEE.bindingEnergy
Description:	Absolute value of the binding energy immediately after CE (erg). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Binding_Energy>CE(1), Binding_Energy>CE(2), Binding_Energy>CE(SN), Binding_Energy>CE(CP)

BINDING_ENERGY_PRE_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.bindingEnergy
Description:	Absolute value of the binding energy at the onset of unstable RLOF leading to the CE (erg). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Binding_Energy<CE(1), Binding_Energy<CE(2), Binding_Energy<CE(SN), Binding_Energy<CE(CP)

CHEMICALLY_HOMOGENEOUS_MAIN_SEQUENCE

Data type:	BOOL
COMPAS variable:	BaseStar::m_CHE
Description:	Flag to indicate whether the star evolved as a CH star for its entire MS lifetime. TRUE indicates star evolved as CH star for entire MS lifetime. FALSE indicates star spun down and switched from CH to a MS_gt_07 star.
Header Strings:	CH_on_MS, CH_on_MS(1), CH_on_MS(2), CH_on_MS(SN), CH_on_MS(CP)

CO_CORE_MASS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_COCoreMass
Description:	Carbon-Oxygen core mass ($M_\odot$).
Header Strings:	Mass_CO_Core, Mass_CO_Core(1), Mass_CO_Core(2), Mass_CO_Core(SN), Mass_CO_Core(CP)

CO_CORE_MASS_AT_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.COCoreMass
Description: Carbon-Oxygen core mass at the onset of unstable RLOF leading to the CE ($M_\odot$).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Mass_CO_Core@CE(1), Mass_CO_Core@CE(2), Mass_CO_Core@CE(SN), Mass_CO_Core@CE(CP)

CO_CORE_MASS_AT_COMPACT_OBJECT_FORMATION

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.COCoreMassAtCOFormation
Description: Carbon-Oxygen core mass immediately prior to a supernova ($M_\odot$).
Header Strings: Mass_CO_Core@CO, Mass_CO_Core@CO(1), Mass_CO_Core@CO(2), Mass_CO_Core@CO(SN), Mass_CO_Core@CO(CP)

CORE_MASS

Data type: DOUBLE
COMPAS variable: BaseStar::m_CoreMass
Description: Core mass ($M_\odot$).
Header Strings: Mass_Core, Mass_Core(1), Mass_Core(2), Mass_Core(SN), Mass_Core(CP)

CORE_MASS_AT_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.CoreMass
Description: Core mass at the onset of unstable RLOF leading to the CE ($M_\odot$).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Mass_Core@CE(1), Mass_Core@CE(2), Mass_Core@CE(SN), Mass_Core@CE(CP)

CORE_MASS_AT_COMPACT_OBJECT_FORMATION

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.CoreMassAtCOFormation
Description: Core mass immediately prior to a supernova ($M_\odot$).
Header Strings: Mass_Core@CO, Mass_Core@CO(1), Mass_Core@CO(2), Mass_Core@CO(SN), Mass_Core@CO(CP)

DRAWN_KICK_MAGNITUDE

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.drawnKickMagnitude
Description: Magnitude of natal kick without accounting for fallback (km s^{-1}).
Supplied by user in grid file or drawn from distribution (default).
This value is used to calculate the actual kick magnitude.
Header Strings: Drawn_Kick_Magnitude, Drawn_Kick_Magnitude(1), Drawn_Kick_Magnitude(2), Drawn_Kick_Magnitude(SN), Drawn_Kick_Magnitude(CP)

DOMINANT_MASS_LOSS_RATE

Data type: INT
COMPAS variable: BaseStar::m_DMLR
Description: Current dominant mass loss rate printed as one of

None	= 0
Nieuwenhuijzen and de Jager	= 1
Kudritzki and Reimers	= 2
Vassiliadis and Wood	= 3
Wolf-Rayet-like (Hamann, Koesterke and de Koter)	= 4
Vink	= 5
Luminous Blue Variable	= 6

Header Strings: Dominant_Mass_Loss_Rate, Dominant_Mass_Loss_Rate(1), Dominant_Mass_Loss_Rate(2), Dominant_Mass_Loss_Rate(SN), Dominant_Mass_Loss_Rate(CP)

DT

Data type: DOUBLE
COMPAS variable: BaseStar::m_Dt
Description: Current timestep (Myr).
Header Strings: dT, dT(1), dT(2), dT(SN), dT(CP)

DYNAMICAL_TIMESCALE

Data type: DOUBLE
COMPAS variable: BaseStar::m_DynamicalTimescale
Description: Dynamical time (Myr).
Header Strings: Tau_Dynamical, Tau_Dynamical(1), Tau_Dynamical(2), Tau_Dynamical(SN), Tau_Dynamical(CP)

DYNAMICAL_TIMESCALE_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.dynamicalTimescale
Description: Dynamical time immediately following common envelope event (Myr).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Tau_Dynamical>CE(1), Tau_Dynamical>CE(2), Tau_Dynamical>CE(SN), Tau_Dynamical>CE(CP)

DYNAMICAL_TIMESCALE_PRE_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.dynamicalTimescale
Description:	Dynamical timescale immediately prior to common envelope event (Myr). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Tau_Dynamical<CE(1), Tau_Dynamical<CE(2), Tau_Dynamical<CE(SN), Tau_Dynamical<CE(CP)

ECCENTRIC_ANOMALY

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_SupernovaDetails.eccentricAnomaly
Description:	Eccentric anomaly calculated using Kepler's equation.
Header Strings:	Eccentric_Anomaly, Eccentric_Anomaly(1), Eccentric_Anomaly(2), Eccentric_Anomaly(SN), Eccentric_Anomaly(CP)

ENV_MASS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_EnvMass
Description:	Envelope mass calculated using Hurley et al. (2000) ($M_\odot$).
Header Strings:	Mass_Env, Mass_Env(1), Mass_Env(2), Mass_Env(SN), Mass_Env(CP)

ERROR

Data type:	INT
COMPAS variable:	*derived from* BaseStar::m_Error
Description:	Error number (if error condition exists, else 0).
Header Strings:	Error, Error(1), Error(2), Error(SN), Error(CP)

EXPERIENCED_CCSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the star exploded as a core-collapse supernova at any time prior to the current timestep.
Header Strings:	Experienced_CCSN, Experienced_CCSN(1), Experienced_CCSN(2), Experienced_CCSN(SN), Experienced_CCSN(CP)

EXPERIENCED_ECSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the star exploded as an electron-capture supernova at any time prior to the current timestep.
Header Strings:	Experienced_ECSN, Experienced_ECSN(1), Experienced_ECSN(2), Experienced_ECSN(SN), Experienced_ECSN(CP)

EXPERIENCED_PISN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the star exploded as an pair-instability supernova at any time prior to the current timestep.
Header Strings:	Experienced_PISN, Experienced_PISN(1), Experienced_PISN(2), Experienced_PISN(SN), Experienced_PISN(CP)

EXPERIENCED_PPISN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the star exploded as a pulsational pair-instability supernova at any time prior to the current timestep.
Header Strings:	Experienced_PPISN, Experienced_PPISN(1), Experienced_PPISN(2), Experienced_PPISN(SN), Experienced_PPISN(CP)

EXPERIENCED_RLOF

Data type:	BOOL
COMPAS variable:	BinaryConstituentStar::m_RLOFDetails.experiencedRLOF
Description:	Flag to indicate whether the star has overflowed its Roche Lobe at any time prior to the current timestep. *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Experienced_RLOF(1), Experienced_RLOF(2), Experienced_RLOF(SN), Experienced_RLOF(CP)

EXPERIENCED_SN_TYPE

Data type: INT

COMPAS variable: *derived from* BaseStar::m_SupernovaDetails.events.past

Description: The type of supernova event experienced by the star prior to the current timestep. Printed as one of

NONE	=	0
CCSN	=	1
ECSN	=	2
PISN	=	4
PPISN	=	8
USSN	=	16

(see Section **Supernova events/states** for explanation).

Header Strings: Experienced_SN_Type, Experienced_SN_Type(1), Experienced_SN_Type(2), Experienced_SN_Type(SN), Experienced_SN_Type(CP)

EXPERIENCED_USSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the star exploded as an ultra-stripped supernova at any time prior to the current timestep.
Header Strings:	Experienced_USSN, Experienced_USSN(1), Experienced_USSN(2), Experienced_USSN(SN), Experienced_USSN(CP)

FALLBACK_FRACTION

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_SupernovaDetails.fallbackFraction
Description:	Fallback fraction during a supernova.
Header Strings:	Fallback_Fraction, Fallback_Fraction(1), Fallback_Fraction(2), Fallback_Fraction(SN), Fallback_Fraction(CP)

HE_CORE_MASS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_HeCoreMass
Description:	Helium core mass ($M_\odot$).
Header Strings:	Mass_He_Core, Mass_He_Core(1), Mass_He_Core(2), Mass_He_Core(SN), Mass_He_Core(CP)

HE_CORE_MASS_AT_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.HeCoreMass
Description:	Helium core mass at the onset of unstable RLOF leading to the CE ($M_\odot$). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Mass_He_Core@CE(1), Mass_He_Core@CE(2), Mass_He_Core@CE(SN), Mass_He_Core@CE(CP)

HE_CORE_MASS_AT_COMPACT_OBJECT_FORMATION

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_SupernovaDetails.HeCoreMassAtCOFormation
Description:	Helium core mass immediately prior to a supernova ($M_\odot$).
Header Strings:	Mass_He_Core@CO, Mass_He_Core@CO(1), Mass_He_Core@CO(2), Mass_He_Core@CO(SN), Mass_He_Core@CO(CP)

ID

Data type:	UNSIGNED LONG INT
COMPAS variable:	BaseStar::m_ObjectId
Description:	Unique object identifier for C++ object – used in debugging to identify objects.
Header Strings:	ID, ID(1), ID(2), ID(SN), ID(CP)

Note that this property has the same header string as BINARY_PROPERTY::ID & BINARY_PROPERTY::RLOF_CURRENT_ID. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

INITIAL_STELLAR_TYPE

Data type:	INT
COMPAS variable:	BaseStar::m_ObjectId
Description:	Stellar type at zero age main-sequence (per Hurley et al. (2000)).
Header Strings:	Stellar_Type@ZAMS, Stellar_Type@ZAMS(1), Stellar_Type@ZAMS(2), Stellar_Type@ZAMS(SN), Stellar_Type@ZAMS(CP)

Note that this property has the same header string as INITIAL_STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

INITIAL_STELLAR_TYPE_NAME

Data type:	STRING
COMPAS variable:	*derived from* BaseStar::m_ObjectId
Description:	Stellar type name (per Hurley et al. (2000)) at zero age main-sequence. e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header Strings:	Stellar_Type@ZAMS, Stellar_Type@ZAMS(1), Stellar_Type@ZAMS(2), Stellar_Type@ZAMS(SN), Stellar_Type@ZAMS(CP)

Note that this property has the same header string as INITIAL_STELLAR_TYPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

IS_CCSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.current
Description:	Flag to indicate whether the star is currently a core-collapse supernova.
Header Strings:	CCSN, CCSN(1), CCSN(2), CCSN(SN), CCSN(CP)

IS_ECSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.current
Description:	Flag to indicate whether the star is currently an electron-capture supernova.
Header Strings:	ECSN, ECSN(1), ECSN(2), ECSN(SN), ECSN(CP)

IS_HYDROGEN_POOR

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.isHydrogenPoor
Description:	Flag to indicate if the star is hydrogen poor.
Header Strings:	Is_Hydrogen_Poor, Is_Hydrogen_Poor(1), Is_Hydrogen_Poor(2), Is_Hydrogen_Poor(SN), Is_Hydrogen_Poor(CP)

IS_PISN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.current
Description:	Flag to indicate whether the star is currently a pair-instability supernova.
Header Strings:	PISN, PISN(1), PISN(2), PISN(SN), PISN(CP)

IS_PPISN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.current
Description:	Flag to indicate whether the star is currently a pulsational pair-instability supernova.
Header Strings:	PPISN, PPISN(1), PPISN(2), PPISN(SN), PPISN(CP)

IS_RLOF

Data type:	BOOL
COMPAS variable:	*derived from* BinaryConstituentStar::m_RLOFDetails.isRLOF
Description:	Flag to indicate whether the star is currently undergoing Roche Lobe overflow. *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	RLOF(1), RLOF(2), RLOF(SN), RLOF(CP)

IS_USSN

Data type:	BOOL
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.current
Description:	Flag to indicate whether the star is currently an ultra-stripped supernova.
Header Strings:	USSN, USSN(1), USSN(2), USSN(SN), USSN(CP)

KICK_MAGNITUDE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_SupernovaDetails.kickMagnitude
Description:	Magnitude of natal kick received during a supernova (km s^{-1}). Calculated using the drawn kick magnitude.
Header Strings:	Applied_Kick_Magnitude, Applied_Kick_Magnitude(1), Applied_Kick_Magnitude(2), Applied_Kick_Magnitude(SN), Applied_Kick_Magnitude(CP)

LAMBDA_AT_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.lambda
Description:	Common-envelope lambda parameter calculated at the unstable RLOF leading to the CE. *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Lambda@CE(1), Lambda@CE(2), Lambda@CE(SN), Lambda@CE(CP)

LAMBDA_DEWI

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.dewi
Description:	Envelope binding energy parameter *lambda* calculated as per Dewi & Tauris (2000) using the fit from Appendix A of Claeys et al. (2014).
Header Strings:	Dewi, Dewi(1), Dewi(2), Dewi(SN), Dewi(CP)

LAMBDA_FIXED

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.fixed
Description:	Universal common envelope lambda parameter specified by the user (program option *--common-envelope-lambda*).
Header Strings:	Lambda_Fixed, Lambda_Fixed(1), Lambda_Fixed(2), Lambda_Fixed(SN), Lambda_Fixed(CP)

LAMBDA_KRUCKOW

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.kruckow
Description:	Envelope binding energy parameter *lambda* calculated as per Kruckow et al. (2016) with the alpha exponent set by program option *--common-envelope-slope-Kruckow*. Spectrum fit to the region bounded by the upper and lower limits as shown in Kruckow et al. (2016, Fig. 1).
Header Strings:	Kruckow, Kruckow(1), Kruckow(2), Kruckow(SN), Kruckow(CP)

LAMBDA_KRUCKOW_BOTTOM

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.kruckowBottom
Description:	Envelope binding energy parameter *lambda* calculated as per Kruckow et al. (2016) with the alpha exponent set to -1. Spectrum fit to the region bounded by the upper and lower limits as shown in Kruckow et al. (2016, Fig. 1).
Header Strings:	Kruckow_Bottom, Kruckow_Bottom(1), Kruckow_Bottom(2), Kruckow_Bottom(SN), Kruckow_Bottom(CP)

LAMBDA_KRUCKOW_MIDDLE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.kruckowMiddle
Description:	Envelope binding energy parameter *lambda* calculated as per Kruckow et al. (2016) with the alpha exponent set to $-\frac{4}{5}$. Spectrum fit to the region bounded by the upper and lower limits as shown in Kruckow et al. (2016, Fig. 1).
Header Strings:	Kruckow_Middle, Kruckow_Middle(1), Kruckow_Middle(2), Kruckow_Middle(SN), Kruckow_Middle(CP)

LAMBDA_KRUCKOW_TOP

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Lambdas.kruckowTop
Description:	Envelope binding energy parameter *lambda* calculated as per Kruckow et al. (2016) with the alpha exponent set to $-\frac{2}{3}$. Spectrum fit to the region bounded by the upper and lower limits as shown in Kruckow et al. (2016, Fig. 1).
Header Strings:	Kruckow_Top, Kruckow_Top(1), Kruckow_Top(2), Kruckow_Top(SN), Kruckow_Top(CP)

LAMBDA_LOVERIDGE

Data type: DOUBLE
COMPAS variable: BaseStar::m_Lambdas.loveridge
Description: Envelope binding energy parameter *lambda* calculated as per Webbink (1984) & Loveridge et al. (2011).
Header Strings: Loveridge, Loveridge(1), Loveridge(2), Loveridge(SN), Loveridge(CP)

LAMBDA_LOVERIDGE_WINDS

Data type: DOUBLE
COMPAS variable: BaseStar::m_Lambdas.loveridgeWinds
Description: Envelope binding energy parameter *lambda* calculated as per Webbink (1984) & Loveridge et al. (2011) including winds.
Header Strings: Loveridge_Winds, Loveridge_Winds(1), Loveridge_Winds(2), Loveridge_Winds(SN), Loveridge_Winds(CP)

LAMBDA_NANJING

Data type: DOUBLE
COMPAS variable: BaseStar::m_Lambdas.nanjing
Description: Envelope binding energy parameter *lambda* calculated as per Xu & Li (2010).
Header Strings: Lambda_Nanjing, Lambda_Nanjing(1), Lambda_Nanjing(2), Lambda_Nanjing(SN), Lambda_Nanjing(CP)

LBV_PHASE_FLAG

Data type: BOOL
COMPAS variable: BaseStar::m_LBVphaseFlag
Description: Flag to indicate if the star ever entered the luminous blue variable phase.
Header Strings: LBV_Phase_Flag, LBV_Phase_Flag(1), LBV_Phase_Flag(2), LBV_Phase_Flag(SN), LBV_Phase_Flag(CP)

LUMINOSITY

Data type: DOUBLE
COMPAS variable: BaseStar::m_Luminosity
Description: Luminosity ($L_\odot$).
Header Strings: Luminosity, Luminosity(1), Luminosity(2), Luminosity(SN), Luminosity(CP)

LUMINOSITY_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.luminosity
Description: Luminosity immediately following common envelope event ($L_\odot$).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Luminosity>CE(1), Luminosity>CE(2), Luminosity>CE(SN), Luminosity>CE(CP)

LUMINOSITY_PRE_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.luminosity
Description:	Luminosity at the onset of unstable RLOF leading to the CE ($L_\odot$). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Luminosity<CE(1), Luminosity<CE(2), Luminosity<CE(SN), Luminosity<CE(CP)

MASS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Mass
Description:	Mass ($M_\odot$).
Header Strings:	Mass, Mass(1), Mass(2), Mass(SN), Mass(CP)

Note that this property has the same header string as RLOF_CURRENT_STAR1_MASS & RLOF_CURRENT_STAR2_MASS. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

MASS_0

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Mass0
Description:	Effective initial mass ($M_\odot$).
Header Strings:	Mass_0, Mass_0(1), Mass_0(2), Mass_0(SN), Mass_0(CP)

MASS_LOSS_DIFF

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_MassLossDiff
Description:	The amount of mass lost due to winds ($M_\odot$). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	dmWinds(1), dmWinds(2), dmWinds(SN), dmWinds(CP)

MASS_TRANSFER_DIFF

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_MassTransferDiff
Description:	The amount of mass accreted or donated during a mass transfer episode ($M_\odot$). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	dmMT(1), dmMT(2), dmMT(SN), dmMT(CP)

MDOT

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Mdot
Description:	Mass loss rate ($M_\odot$ yr^{-1}).
Header Strings:	Mdot, Mdot(1), Mdot(2), Mdot(SN), Mdot(CP)

MEAN_ANOMALY

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_SupernovaDetails.meanAnomaly
Description:	Mean anomaly of supernova kick. Supplied by user in grid file, default = random number drawn from [0..2π). See https://en.wikipedia.org/wiki/Mean_anomaly for explanation.
Header Strings:	SN_Kick_Mean_Anomaly, SN_Kick_Mean_Anomaly(1), SN_Kick_Mean_Anomaly(2), SN_Kick_Mean_Anomaly(SN), SN_Kick_Mean_Anomaly(CP)

METALLICITY

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Metallicity
Description:	ZAMS Metallicity.
Header Strings:	Metallicity@ZAMS, Metallicity@ZAMS(1), Metallicity@ZAMS(2), Metallicity@ZAMS(SN), Metallicity@ZAMS(CP)

MT_DONOR_HIST

Data type:	STRING
COMPAS variable:	BaseStar::m_MassTransferDonorHistory
Description:	A list of all of the stellar types from which the current star was a Mass Transfer donor. This can be readily converted into the different cases of Mass Transfer, depending on the working definition. The output string is formatted as #-#-#... where each # represents a Hurley stellar type, delimited by dashes, in chronological order. E.g, 2-8 means the star was a MT donor as a HG(2) star, and later as a HeHG(8) star.
Header Strings:	MT_Donor_Hist, MT_Donor_Hist(1), MT_Donor_Hist(2), MT_Donor_Hist(SN), MT_Donor_Hist(CP)

MZAMS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_MZAMS
Description:	ZAMS Mass ($M_\odot$).
Header Strings:	Mass@ZAMS, Mass@ZAMS(1), Mass@ZAMS(2), Mass@ZAMS(SN), Mass@ZAMS(CP)

NUCLEAR_TIMESCALE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_NuclearTimescale
Description:	Nuclear timescale (Myr).
Header Strings:	Tau_Nuclear, Tau_Nuclear(1), Tau_Nuclear(2), Tau_Nuclear(SN), Tau_Nuclear(CP)

NUCLEAR_TIMESCALE_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.nuclearTimescale
Description: Nuclear timescale immediately following common envelope event (Myr).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Tau_Nuclear>CE(1), Tau_Nuclear>CE(2), Tau_Nuclear>CE(SN), Tau_Nuclear>CE(CP)

NUCLEAR_TIMESCALE_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.nuclearTimescale
Description: Nuclear timescale at the onset of unstable RLOF leading to the CE (Myr).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Tau_Nuclear<CE(1), Tau_Nuclear<CE(2), Tau_Nuclear<CE(SN), Tau_Nuclear<CE(CP)

OMEGA

Data type: DOUBLE
COMPAS variable: BaseStar::m_Omega
Description: Angular frequency (yr^{-1}).
Header Strings: Omega, Omega(1), Omega(2), Omega(SN), Omega(CP)

OMEGA_BREAK

Data type: DOUBLE
COMPAS variable: BaseStar::m_OmegaBreak
Description: Break-up angular frequency (yr^{-1}).
Header Strings: Omega_Break, Omega_Break(1), Omega_Break(2), Omega_Break(SN), Omega_Break(CP)

OMEGA_ZAMS

Data type: DOUBLE
COMPAS variable: BaseStar::m_OmegaZAMS
Description: Angular frequency at ZAMS (yr^{-1}).
Header Strings: Omega@ZAMS, Omega@ZAMS(1), Omega@ZAMS(2), Omega@ZAMS(SN), Omega@ZAMS(CP)

ORBITAL_ENERGY_POST_SUPERNOVA

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_PostSNeOrbitalEnergy
Description: Absolute value of orbital energy immediately following supernova event ($M_\odot$ AU^2 yr^{-2}).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Orbital_Energy>SN(1), Orbital_Energy>SN(2), Orbital_Energy>SN(SN), Orbital_Energy>SN(CP)

ORBITAL_ENERGY_PRE_SUPERNOVA

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_PreSNeOrbitalEnergy
Description:	Orbital energy immediately prior to supernova event ($M_\odot$ AU^2 yr^{-2}). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Orbital_Energy<SN(1), Orbital_Energy<SN(2), Orbital_Energy<SN(SN), Orbital_Energy<SN(CP)

PULSAR_MAGNETIC_FIELD

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_PulsarDetails.magneticField
Description:	Pulsar magnetic field strength (G).
Header Strings:	Pulsar_Mag_Field, Pulsar_Mag_Field(1), Pulsar_Mag_Field(2), Pulsar_Mag_Field(SN), Pulsar_Mag_Field(CP)

PULSAR_SPIN_DOWN_RATE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_PulsarDetails.spinDownRate
Description:	Pulsar spin-down rate.
Header Strings:	Pulsar_Spin_Down, Pulsar_Spin_Down(1), Pulsar_Spin_Down(2), Pulsar_Spin_Down(SN), Pulsar_Spin_Down(CP)

PULSAR_SPIN_FREQUENCY

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_PulsarDetails.spinFrequency
Description:	Pulsar spin angular frequency (rads s^{-1}).
Header Strings:	Pulsar_Spin_Freq, Pulsar_Spin_Freq(1), Pulsar_Spin_Freq(2), Pulsar_Spin_Freq(SN), Pulsar_Spin_Freq(CP)

PULSAR_SPIN_PERIOD

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_PulsarDetails.spinPeriod
Description:	Pulsar spin period (ms).
Header Strings:	Pulsar_Spin_Period, Pulsar_Spin_Period(1), Pulsar_Spin_Period(2), Pulsar_Spin_Period(SN), Pulsar_Spin_Period(CP)

RADIAL_EXPANSION_TIMESCALE

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_RadialExpansionTimescale
Description:	e-folding time of stellar radius (Myr).
Header Strings:	Tau_Radial, Tau_Radial(1), Tau_Radial(2), Tau_Radial(SN), Tau_Radial(CP)

RADIAL_EXPANSION_TIMESCALE_POST_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.postCEE.radialExpansionTimescale
Description:	e-folding time of stellar radius immediately following common envelope event (Myr). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Tau_Radial<CE(1), Tau_Radial<CE(2), Tau_Radial<CE(SN), Tau_Radial<CE(CP)

RADIAL_EXPANSION_TIMESCALE_PRE_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.radialExpansionTimescale
Description:	e-folding time of stellar radius at the onset of unstable RLOF leading to the CE (Myr). *Applies only to constituent stars of a binary system (i.e. does not apply to SSE).*
Header Strings:	Tau_Radial<CE(1), Tau_Radial<CE(2), Tau_Radial<CE(SN), Tau_Radial<CE(CP)

RADIUS

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_Radius
Description:	Radius ($R_{\odot}$).
Header Strings:	Radius, Radius(1), Radius(2), Radius(SN), Radius(CP)

Note that this property has the same header string as RLOF_CURRENT_STAR1_RADIUS & RLOF_CURRENT_STAR2_RADIUS. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RANDOM_SEED

Data type:	DOUBLE
COMPAS variable:	BaseStar::m_RandomSeed
Description:	Seed for random number generator for this star.
Header Strings:	SEED, SEED(1), SEED(2), SEED(SN), SEED(CP)

Note that this property has the same header string as BINARY_PROPERTY::RANDOM_SEED & BINARY_PROPERTY::RLOF_CURRENT_RANDOM_SEED. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RECYCLED_NEUTRON_STAR

Data type:	DOUBLE
COMPAS variable:	*derived from* BaseStar::m_SupernovaDetails.events.past
Description:	Flag to indicate whether the object was a recycled neutron star at any time prior to the current timestep (was a neutron star accreting mass).
Header Strings:	Recycled_NS, Recycled_NS(1), Recycled_NS(2), Recycled_NS(SN), Recycled_NS(CP)

RLOF_ONTO_NS

Data type: DOUBLE
COMPAS variable: *derived from* BaseStar::m_SupernovaDetails.events.past
Description: Flag to indicate whether the star transferred mass to a neutron star at any time prior to the current timestep.
Header Strings: RLOF→NS, RLOF→NS(1), RLOF→NS(2), RLOF→NS(SN), RLOF→NS(CP)

RUNAWAY

Data type: DOUBLE
COMPAS variable: *derived from* BaseStar::m_SupernovaDetails.events.past
Description: Flag to indicate whether the star was unbound by a supernova event at any time prior to the current timestep. (i.e Unbound after supernova event and not a WD, NS, BH or MR).
Header Strings: Runaway, Runaway(1), Runaway(2), Runaway(SN), Runaway(CP)

RZAMS

Data type: DOUBLE
COMPAS variable: BaseStar::m_RZAMS
Description: ZAMS Radius ($R_\odot$).
Header Strings: R@ZAMS, R@ZAMS(1), R@ZAMS(2), R@ZAMS(SN), R@ZAMS(CP)

SN_TYPE

Data type: INT
COMPAS variable: *derived from* BaseStar::m_SupernovaDetails.events.current
Description: The type of supernova event currently being experienced by the star. Printed as one of

NONE	=	0
CCSN	=	1
ECSN	=	2
PISN	=	4
PPISN	=	8
USSN	=	16

(see section **Supernova events/states** for explanation).
Header Strings: SN_Type, SN_Type(1), SN_Type(2), SN_Type(SN), SN_Type(CP)

STELLAR_TYPE

Data type:	INT
COMPAS variable:	BaseStar::m_StellarType
Description:	Stellar type (per Hurley et al. (2000)).
Header Strings:	Stellar_Type, Stellar_Type(1), Stellar_Type(2), Stellar_Type(SN), Stellar_Type(CP)

Note that this property has the same header string as STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_NAME

Data type:	STRING
COMPAS variable:	*derived from* BaseStar::m_StellarType
Description:	Stellar type name (per Hurley et al. (2000)). e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header Strings:	Stellar_Type, Stellar_Type(1), Stellar_Type(2), Stellar_Type(SN), Stellar_Type(CP)

Note that this property has the same header string as STELLAR_TYPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_PREV

Data type:	INT
COMPAS variable:	BaseStar::m_StellarTypePrev
Description:	Stellar type (per Hurley et al. (2000)) at previous timestep.
Header Strings:	Stellar_Type_Prev, Stellar_Type_Prev(1), Stellar_Type_Prev(2), Stellar_Type_Prev(SN), Stellar_Type_Prev(CP)

Note that this property has the same header string as STELLAR_TYPE_PREV_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_PREV_NAME

Data type:	STRING
COMPAS variable:	*derived from* BaseStar::m_StellarTypePrev
Description:	Stellar type name (per Hurley et al. (2000)) at previous timestep. e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header Strings:	Stellar_Type_Prev, Stellar_Type_Prev(1), Stellar_Type_Prev(2), Stellar_Type_Prev(SN), Stellar_Type_Prev(CP)

Note that this property has the same header string as STELLAR_TYPE_PREV. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SUPERNOVA_KICK_MAGNITUDE_MAGNITUDE_RANDOM_NUMBER

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.kickMagnitudeRandom
Description: Random number for drawing the supernova kick magnitude (if required).
Supplied by user in grid file, default = random number drawn from [0..1).
Header Strings: SN_Kick_Magnitude_Random_Number, SN_Kick_Magnitude_Random_Number(1), SN_Kick_Magnitude_Random_Number(2), SN_Kick_Magnitude_Random_Number(SN), SN_Kick_Magnitude_Random_Number(CP)

SUPERNOVA_PHI

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.phi
Description: Angle between 'x' and 'y', both in the orbital plane of supernovae vector (rad).
Supplied by user in grid file, default = random number drawn from [0..2pi).
Header Strings: SN_Kick_Phi, SN_Kick_Phi(1), SN_Kick_Phi(2), SN_Kick_Phi(SN), SN_Kick_Phi(CP)

SUPERNOVA_THETA

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.theta
Description: Angle between the orbital plane and the 'z' axis of supernovae vector (rad).
Supplied by user in grid file, default = drawn from distribution specified by program option *--kick_direction*.
Header Strings: SN_Kick_Theta, SN_Kick_Theta(1), SN_Kick_Theta(2), SN_Kick_Theta(SN), SN_Kick_Theta(CP)

TEMPERATURE

Data type: DOUBLE
COMPAS variable: BaseStar::m_Temperature
Description: Effective temperature (K).
Header Strings: Teff, Teff(1), Teff(2), Teff(SN), Teff(CP)

TEMPERATURE_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.temperature
Description: Effective temperature immediately following common envelope event (K).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Teff>CE(1), Teff>CE(2), Teff>CE(SN), Teff>CE(CP)

TEMPERATURE_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.temperature
Description: Effective temperature at the unstable RLOF leading to the CE (K).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Teff<CE(1), Teff<CE(2), Teff<CE(SN), Teff<CE(CP)

THERMAL_TIMESCALE

Data type: DOUBLE
COMPAS variable: BaseStar::m_ThermalTimescale
Description: Thermal timescale (Myr).
Header Strings: Tau_Thermal, Tau_Thermal(1), Tau_Thermal(2), Tau_Thermal(SN), Tau_Thermal(CP)

THERMAL_TIMESCALE_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.thermalTimescale
Description: Thermal timescale immediately following common envelope event (Myr).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Tau_Thermal>CE(1), Tau_Thermal>CE(2), Tau_Thermal>CE(SN), Tau_Thermal>CE(CP)

THERMAL_TIMESCALE_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.thermalTimescale
Description: Thermal timescale at the onset of the unstable RLOF leading to the CE (Myr).
Applies only to constituent stars of a binary system (i.e. does not apply to SSE).
Header Strings: Tau_Thermal<CE(1), Tau_Thermal<CE(2), Tau_Thermal<CE(SN), Tau_Thermal<CE(CP)

TIME

Data type: DOUBLE
COMPAS variable: BaseStar::m_Time
Description: Time since ZAMS (Myr).
Header Strings: Time, Time(1), Time(2), Time(SN), Time(CP)

Note that this property has the same header string as BINARY_PROPERTY::TIME & BINARY_PROPERTY::RLOF_CURRENT_TIME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

TIMESCALE_MS

Data type: DOUBLE
COMPAS variable: BaseStar::m_Timescales[tMS]
Description: Main Sequence timescale (Myr).
Header Strings: tMS, tMS(1), tMS(2), tMS(SN), tMS(CP)

TOTAL_MASS_AT_COMPACT_OBJECT_FORMATION

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.totalMassAtCOFormation
Description: Total mass of the star at the beginning of a supernova event ($M_\odot$).
Header Strings: Mass_Total@CO, Mass_Total@CO(1), Mass_Total@CO(2), Mass_Total@CO(SN), Mass_Total@CO(CP)

TRUE_ANOMALY

Data type: DOUBLE
COMPAS variable: BaseStar::m_SupernovaDetails.trueAnomaly
Description: True anomaly calculated using Kepler's equation (rad).
See https://en.wikipedia.org/wiki/True_anomaly for explanation.
Header Strings: True_Anomaly(psi), True_Anomaly(psi)(1), True_Anomaly(psi)(2), True_Anomaly(psi)(SN), True_Anomaly(psi)(CP)

ZETA_HURLEY

Data type: DOUBLE
COMPAS variable: BaseStar::m_Zetas.hurley
Description: Adiabatic exponent calculated per Hurley et al. (2000) using core mass.
Header Strings: Zeta_Hurley, Zeta_Hurley(1), Zeta_Hurley(2), Zeta_Hurley(SN), Zeta_Hurley(CP)

ZETA_HURLEY_HE

Data type: DOUBLE
COMPAS variable: BaseStar::m_Zetas.hurleyHe
Description: Adiabatic exponent calculated per Hurley et al. (2000) using He core mass.
Header Strings: Zeta_Hurley_He, Zeta_Hurley_He(1), Zeta_Hurley_He(2), Zeta_Hurley_He(SN), Zeta_Hurley_He(CP)

ZETA_SOBERMAN

Data type: DOUBLE
COMPAS variable: BaseStar::m_Zetas.soberman
Description: Adiabatic exponent calculated per Soberman et al. (1997) using core mass.
Header Strings: Zeta_Soberman, Zeta_Soberman(1), Zeta_Soberman(2), Zeta_Soberman(SN), Zeta_Soberman(CP)

ZETA_SOBERMAN_HE

Data type: DOUBLE
COMPAS variable: BaseStar::m_Zetas.sobermanHe
Description: Adiabatic exponent calculated per Soberman et al. (1997) using He core mass.
Header Strings: Zeta_Soberman_He, Zeta_Soberman_He(1), Zeta_Soberman_He(2), Zeta_Soberman_He(SN), Zeta_Soberman_He(CP)

Supernova events/states

Supernova events/states, both current ("is") and past ("experienced"), are stored within COMPAS as bit maps. That means different values can be ORed or ANDed into the bit map, so that various events or states can be set concurrently.

The values shown below for the SN_EVENT type are powers of 2 so that they can be used in a bit map and manipulated with bit-wise logical operators. Any of the individual supernova event/state types that make up the SN_EVENT type can be set independently of any other event/state.

```
enum class SN_EVENT: int {
    NONE          =  0,
    CCSN          =  1,
    ECSN          =  2,
    PISN          =  4,
    PPISN         =  8,
    USSN          =  16,
    RUNAWAY       =  32,
    RECYCLED_NS   =  64,
    RLOF_ONTO_NS  =  128
};

const COMPASUnorderedMap<SN_EVENT, std::string> SN_EVENT_LABEL = {
    { SN_EVENT::NONE,          "No Supernova" },
    { SN_EVENT::CCSN,          "Core Collapse Supernova" },
    { SN_EVENT::ECSN,          "Electron Capture Supernova" },
    { SN_EVENT::PISN,          "Pair Instability Supernova" },
    { SN_EVENT::PPISN,         "Pulsational Pair Instability Supernova" },
    { SN_EVENT::USSN,          "Ultra Stripped Supernova" },
    { SN_EVENT::RUNAWAY,       "Runaway Companion" },
    { SN_EVENT::RECYCLED_NS,   "Recycled Neutron Star" },
    { SN_EVENT::RLOF_ONTO_NS,  "Donated Mass to Neutron Star through RLOF" }
};
```

A convenience function (shown below) is provided in **utils.cpp** to interpret the bit map.

```
/*
* Returns a single SN type based on the SN_EVENT parameter passed
*
* Returns (in priority order):
*
* SN_EVENT::CCSN   iff CCSN bit is set and USSN bit is not set
* SN_EVENT::ECSN   iff ECSN bit is set
* SN_EVENT::PISN   iff PISN bit is set
* SN_EVENT::PPISN  iff PPISN bit is set
* SN_EVENT::USSN   iff USSN bit is set
* SN_EVENT::NONE   otherwise
*
*
* @param   [IN]   p_SNEvent      SN_EVENT mask to check for SN event type
* @return                        SN_EVENT
*/
SN_EVENT SNEventType(const SN_EVENT p_SNEvent) {

    if ((p_SNEvent & (SN_EVENT::CCSN | SN_EVENT::USSN)) == SN_EVENT::CCSN ) return SN_EVENT::CCSN;
    if ((p_SNEvent & SN_EVENT::ECSN )                   == SN_EVENT::ECSN ) return SN_EVENT::ECSN;
    if ((p_SNEvent & SN_EVENT::PISN )                   == SN_EVENT::PISN ) return SN_EVENT::PISN;
    if ((p_SNEvent & SN_EVENT::PPISN)                   == SN_EVENT::PPISN) return SN_EVENT::PPISN;
    if ((p_SNEvent & SN_EVENT::USSN )                   == SN_EVENT::USSN ) return SN_EVENT::USSN;

    return SN_EVENT::NONE;
}
```

Log File Record Specification: Binary Properties

As described in Section **Extended Logging**, when specifying known properties in a log file record specification record, the property name must be prefixed with the property type.

Currently there is a single binary property type available for use: BINARY_PROPERTY.

For example, to specify the property SEMI_MAJOR_AXIS_PRE_COMMON_ENVELOPE for a binary star being evolved in BSE, use:

BINARY_PROPERTY::SEMI_MAJOR_AXIS_PRE_COMMON_ENVELOPE

Following is the list of binary properties available for inclusion in log file record specifiers.

Binary Properties

CIRCULARIZATION_TIMESCALE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_CircularizationTimescale
Description:	Tidal circularisation timescale (Myr)
Header String:	Tau_Circ

COMMON_ENVELOPE_AT_LEAST_ONCE

Data type:	BOOL
COMPAS variable:	*derived from* BaseBinaryStar::m_CEDetails.CEEcount
Description:	Flag to indicate if there has been at least one common envelope event.
Header String:	CEE

Note that this property has the same header string as RLOF_CURRENT_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

COMMON_ENVELOPE_EVENT_COUNT

Data type:	INT
COMPAS variable:	BaseBinaryStar::m_CEDetails.CEEcount
Description:	The number of common envelope events.
Header String:	CE_Event_Count

DIMENSIONLESS_KICK_MAGNITUDE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_uK
Description:	Dimensionless kick magnitude supplied by user (see option *--fix-dimensionless-kick-magnitude)*.
Header String:	Kick_Magnitude(uK)

DOUBLE_CORE_COMMON_ENVELOPE

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_CEDetails.doubleCoreCE
Description:	Flag to indicate double-core common envelope.
Header String:	Double_Core_CE

DT

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_Dt
Description:	Current timestep (Myr).
Header String:	dT

ECCENTRICITY

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_Eccentricity
Description:	Orbital eccentricity.
Header String:	Eccentricity

ECCENTRICITY_AT_DCO_FORMATION

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_EccentricityAtDCOFormation
Description:	Orbital eccentricity at DCO formation.
Header String:	Eccentricity@DCO

ECCENTRICITY_INITIAL

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_EccentricityInitial
Description:	Supplied by user via grid file or sampled from distribution (see *--eccentricity-distribution option*) (default).
Header String:	Eccentricity@ZAMS

ECCENTRICITY_POST_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_CEDetails.postCEE.eccentricity
Description:	Eccentricity immediately following common envelope event.
Header String:	Eccentricity>CE

ECCENTRICITY_PRE_SUPERNOVA

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_EccentricityPreSN
Description:	Eccentricity of the binary immediately prior to supernova event.
Header String:	Eccentricity<SN

ECCENTRICITY_PRE_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_CEDetails.preCEE.eccentricity
Description:	Eccentricity at the onset of RLOF leading to the CE.
Header String:	Eccentricity<CE

ERROR

Data type:	INT
COMPAS variable:	*derived from* BaseBinaryStar::m_Error
Description:	Error number (if error condition exists, else 0).
Header String:	Error

ID

Data type:	INT
COMPAS variable:	BaseBinaryStar::m_ObjectId
Description:	Unique object identifier for C++ object – used in debugging to identify objects.
Header String:	ID

Note that this property has the same header string as ANY_STAR_PROPERTY::ID & RLOF_CURRENT_ID. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

IMMEDIATE_RLOF_POST_COMMON_ENVELOPE

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.immediateRLOFPostCEE
Description:	Flag to indicate if either star overflows its Roche lobe immediately following common envelope event.
Header String:	Immediate_RLOF>CE

MASS_1_FINAL

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_Mass1Final
Description:	Mass of the primary star after losing its envelope (assumes complete loss of envelope) ($M_\odot$).
Header String:	Core_Mass_1

MASS_1_POST_COMMON_ENVELOPE

Data type:	DOUBLE
COMPAS variable:	BinaryConstituentStar::m_CEDetails.postCEE.mass
Description:	Mass of the primary star immediately following common envelope event ($M_\odot$).
Header String:	Mass_1>CE

MASS_1_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.mass
Description: Mass of the primary star immediately prior to common envelope event ($M_{\odot}$).
Header String: Mass_1<CE

MASS_2_FINAL

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_Mass2Final
Description: Mass of the secondary star after losing its envelope (assumes complete loss of envelope) ($M_{\odot}$).
Header String: Core_Mass_2

MASS_2_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.mass
Description: Mass of the secondary star immediately following common envelope event ($M_{\odot}$).
Header String: Mass_2>CE

MASS_2_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.mass
Description: Mass of the secondary star immediately prior to common envelope event ($M_{\odot}$).
Header String: Mass_2<CE

MASS_ENV_1

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_MassEnv1
Description: Envelope mass of the primary star ($M_{\odot}$).
Header String: Mass_Env_1

MASS_ENV_2

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_MassEnv2
Description: Envelope mass of the secondary star ($M_{\odot}$).
Header String: Mass_Env_2

MASSES_EQUILIBRATED

Data type: BOOL
COMPAS variable: BaseBinaryStar::m_MassesEquilibrated
Description: Flag to indicate whether chemically homogeneous stars had masses equilibrated and orbit circularised due to Roche lobe overflow during evolution.
Header String: Equilibrated

MASSES_EQUILIBRATED_AT_BIRTH

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_MassesEquilibratedAtBirth
Description:	Flag to indicate whether stars had masses equilibrated and orbit circularised at birth due to Roche lobe overflow.
Header String:	Equilibrated_At_Birth

MASS_TRANSFER_TRACKER_HISTORY

Data type:	INT
COMPAS variable:	*derived from* BaseBinaryStar::m_MassTransferTrackerHistory
Description:	Indicator of mass transfer history for the binary. Will be printed as one of:

NO_MASS_TRANSFER	=	0
STABLE_FROM_1_TO_2	=	1
STABLE_FROM_2_TO_1	=	2
CE_FROM_1_TO_2	=	3
CE_FROM_2_TO_1	=	4
CE_DOUBLE_CORE	=	5
CE_BOTH_MS	=	6
CE_MS_WITH_CO	=	7

Header String:	MT_History

MERGES_IN_HUBBLE_TIME

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_MergesInHubbleTime
Description:	Flag to indicate if the binary compact remnants merge within a Hubble time.
Header String:	Merges_Hubble_Time

OPTIMISTIC_COMMON_ENVELOPE

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_CEDetails.optimisticCE
Description:	Flag that returns TRUE if we have a Hertzsprung-gap star, and we allow it to survive the CE.
Header String:	Optimistic_CE

ORBITAL_VELOCITY

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_OrbitalVelocity
Description:	Orbital velocity (km s^{-1}).
Header String:	Orbital_Velocity

ORBITAL_VELOCITY_PRE_SUPERNOVA

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_OrbitalVelocityPreSN
Description: Orbital velocity immediately prior to supernova event (km s^{-1}).
Header String: Orbital_Velocity<SN

RADIUS_1_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.radius
Description: Radius of the primary star immediately following common envelope event ($R_\odot$).
Header String: Radius_1>CE

RADIUS_1_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.radius
Description: Radius of the primary star at the onset of RLOF leading to the common-envelope episode ($R_\odot$).
Header String: Radius_1<CE

RADIUS_2_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.radius
Description: Radius of the secondary star immediately following common envelope event ($R_\odot$).
Header String: Radius_2>CE

RADIUS_2_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_CEDetails.preCEE.radius
Description: Radius of the secondary star at the onset of RLOF leading to the common-envelope episode ($R_\odot$).
Header String: Radius_2<CE

RANDOM_SEED

Data type: UNSIGNED LONG
COMPAS variable: BaseBinaryStar::m_RandomSeed
Description: Seed for random number generator for this binary star. Optionally supplied by user via option *--random-seed*; default generated from system time.
Header String: SEED

RLOF_CURRENT_COMMON_ENVELOPE

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→isCE
Description:	Flag to indicate if the RLOF leads to a common-envelope event
Header String:	CEE

Note that this property has the same header string as COMMON_ENVELOPE_AT_LEAST_ONCE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_ECCENTRICITY

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→eccentricity
Description:	Eccentricity immediately after RLOF.
Header String:	Eccentricity

RLOF_CURRENT_EVENT_COUNTER

Data type:	UNSIGNED INT
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→eventCounter
Description:	The number of times the binary has overflowed its Roche lobe up to and including this episode
Header String:	Event_Counter

RLOF_CURRENT_SEMI_MAJOR_AXIS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→semiMajorAxis
Description:	Semi-major Axis($R_\odot$) immediately after RLOF.
Header String:	SemiMajorAxis

Note that this property has the same header string as SEMI_MAJOR_AXIS, SEMI_MAJOR_AXIS_RSOL. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR1_MASS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→mass1
Description:	Mass ($M_\odot$) of the primary immediately after RLOF.
Header String:	Mass_1

Note that this property has the same header string as ANY_STAR_PROPERTY::MASS (Star 1). *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR2_MASS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→mass2
Description:	Mass ($M_\odot$) of the secondary immediately after RLOF.
Header String:	Mass_2

Note that this property has the same header string as ANY_STAR_PROPERTY::MASS (Star 2). *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR1_RADIUS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→radius1
Description:	Radius ($R_\odot$) of the primary immediately after RLOF.
Header String:	Radius_1

Note that this property has the same header string as ANY_STAR_PROPERTY::RADIUS (Star 1). *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR2_RADIUS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→radius2
Description:	Radius ($R_\odot$) of the secondary immediately after RLOF.
Header String:	Radius_2

Note that this property has the same header string as ANY_STAR_PROPERTY::RADIUS (Star 2). *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR1_RLOF

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→isRLOF1
Description:	Flag to indicate whether the primary is overflowing its Roche Lobe.
Header String:	RLOF_1

RLOF_CURRENT_STAR2_RLOF

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→isRLOF2
Description:	Flag to indicate whether the secondary is overflowing its Roche Lobe.
Header String:	RLOF_2

RLOF_CURRENT_STAR1_STELLAR_TYPE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→stellarType1
Description:	Stellar type (per Hurley et al. (2000)) of the primary star immediately after RLOF.
Header String:	Type_1

Note that this property has the same header string as RLOF_CURRENT_STAR1_STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_STAR2_STELLAR_TYPE

Data type:	DOUBLE
COMPAS variable:	*derived from* BaseBinaryStar::m_RLOFDetails.currentProps→stellarType1
Description:	Stellar type (per Hurley et al. (2000)) of the secondary star immediately after RLOF.
Header String:	Type_2

Note that this property has the same header string as RLOF_CURRENT_STAR2_STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_CURRENT_TIME

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.currentProps→time
Description:	Time since ZAMS (Myr) immediately after RLOF.
Header String:	Time

Note that this property has the same header string as TIME & ANY_STAR_PROPERTY::TIME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_PREVIOUS_ECCENTRICITY

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→eccentricity
Description:	Eccentricity at the onset of RLOF.
Header String:	Eccentricity_Prev

RLOF_PREVIOUS_EVENT_COUNTER

Data type:	UNSIGNED INT
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→eventCounter
Description:	The number of times the binary has overflowed its Roche lobe up to and excluding this episode
Header String:	Event_Counter_Prev

RLOF_PREVIOUS_SEMI_MAJOR_AXIS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→semiMajorAxis
Description:	Semi-major Axis ($R_{\odot}$) at the onset of RLOF.
Header String:	SemiMajorAxis_Prev

Note that this property has the same header string as SEMI_MAJOR_AXIS, SEMI_MAJOR_AXIS_RSOL. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_PREVIOUS_STAR1_MASS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→mass1
Description:	Mass ($M_{\odot}$) of the primary at the onset of RLOF.
Header String:	Mass_1_Prev

RLOF_PREVIOUS_STAR2_MASS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→mass2
Description:	Mass ($M_{\odot}$) of the secondary at the onset of RLOF.
Header String:	Mass_2_Prev

RLOF_PREVIOUS_STAR1_RADIUS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→radius1
Description:	Radius ($R_{\odot}$) of the primary at the onset of RLOF.
Header String:	Radius_1_Prev

RLOF_PREVIOUS_STAR2_RADIUS

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→radius2
Description:	Radius ($R_{\odot}$) of the secondary at the onset of RLOF.
Header String:	Radius_2_Prev

RLOF_PREVIOUS_STAR1_RLOF

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→isRLOF1
Description:	Flag to indicate whether the primary is overflowing its Roche Lobe.
Header String:	RLOF_1_Prev

RLOF_PREVIOUS_STAR2_RLOF

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_RLOFDetails.previousProps→isRLOF2
Description:	Flag to indicate whether the secondary is overflowing its Roche Lobe.
Header String:	RLOF_2_Prev

RLOF_PREVIOUS_STAR1_STELLAR_TYPE

Data type: INT
COMPAS variable: BaseBinaryStar::m_RLOFDetails.previousProps→stellarType1
Description: Stellar type (per Hurley et al. (2000)) of the primary star at the onset of RLOF.
Header String: Type_1_Prev

Note that this property has the same header string as RLOF_PREVIOUS_STAR1_STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_PREVIOUS_STAR2_STELLAR_TYPE

Data type: INT
COMPAS variable: BaseBinaryStar::m_RLOFDetails.previousProps→stellarType2
Description: Stellar type (per Hurley et al. (2000)) of the secondary star at the onset of RLOF.
Header String: Type_2_Prev

Note that this property has the same header string as RLOF_PREVIOUS_STAR2_STELLAR_TYPE_NAME. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

RLOF_PREVIOUS_TIME

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_RLOFDetails.previousProps→time
Description: Time since ZAMS (Myr) at the onset of RLOF.
Header String: Time_Prev

ROCHE_LOBE_RADIUS_1

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_RocheLobeRadius
Description: Roche radius of the primary star ($R_\odot$).
Header String: RocheLobe(1)|a

ROCHE_LOBE_RADIUS_2

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_RocheLobeRadius
Description: Roche radius of the secondary star ($R_\odot$).
Header String: RocheLobe(2)|a

ROCHE_LOBE_RADIUS_1_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.postCEE.rocheLobe1to2
Description: Roche radius of the primary star immediately following common envelope event ($R_\odot$).
Header String: RocheLobe_1>CE

ROCHE_LOBE_RADIUS_2_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.postCEE.rocheLobe2to1
Description: Roche radius of the secondary star immediately following common envelope event ($R_\odot$).
Header String: RocheLobe_2>CE

ROCHE_LOBE_RADIUS_1_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.preCEE.rocheLobe1to2
Description: Roche radius of the primary star at the onset of RLOF leading to the common-envelope episode ($R_\odot$).
Header String: RocheLobe_1<CE

ROCHE_LOBE_RADIUS_2_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.preCEE.rocheLobe2to1
Description: Roche radius of the secondary star at the onset of RLOF leading to the common-envelope episode ($R_\odot$).
Header String: RocheLobe_2<CE

ROCHE_LOBE_TRACKER_1

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_RocheLobeTracker
Description: Ratio of the primary star's stellar radius to Roche radius (R/RL), evaluated at periapsis.
Header String: Radius|RL

ROCHE_LOBE_TRACKER_2

Data type: DOUBLE
COMPAS variable: BinaryConstituentStar::m_RocheLobeTracker
Description: Ratio of the secondary star's stellar radius to Roche radius (R/RL), evaluated at periapsis.
Header String: Radius|RL

SEMI_MAJOR_AXIS_AT_DCO_FORMATION

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_SemiMajorAxisAtDCOFormation
Description: Semi-major axis at DCO formation (AU).
Header String: Separation@DCO

SEMI_MAJOR_AXIS_INITIAL

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_SemiMajorAxisInitial
Description: Semi-major axis at ZAMS (AU).
Header String: Separation@ZAMS

SEMI_MAJOR_AXIS_POST_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.postCEE.semiMajorAxis
Description: Semi-major axis immediately following common envelope event ($R_\odot$).
Header String: Separation>CE

SEMI_MAJOR_AXIS_PRE_SUPERNOVA

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_SemiMajorAxisPreSN
Description: Semi-major axis immediately prior to supernova event (AU).
Header String: Separation<SN

Note that this property has the same header string as SEMI_MAJOR_AXIS_PRE_SUPERNOVA_RSOL. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SEMI_MAJOR_AXIS_PRE_SUPERNOVA_RSOL

Data type: DOUBLE
COMPAS variable: *derived from* BaseBinaryStar::m_SemiMajorAxisPreSN
Description: Semi-major axis immediately prior to supernova event ($R_\odot$).
Header String: Separation<SN

Note that this property has the same header string as SEMI_MAJOR_AXIS_PRE_SUPERNOVA. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SEMI_MAJOR_AXIS_PRE_COMMON_ENVELOPE

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_CEDetails.preCEE.semiMajorAxis
Description: Semi-major axis at the onset of RLOF leading to the common-envelope episode ($R_\odot$).
Header String: Separation<CE

SEMI_MAJOR_AXIS

Data type: DOUBLE
COMPAS variable: BaseBinaryStar::m_SemiMajorAxis
Description: Semi-major axis at ZAMS (AU).
Header String: Separation@ZAMS

Note that this property has the same header string as SEMI_MAJOR_AXIS_RSOL. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SEMI_MAJOR_AXIS_RSOL

Data type: DOUBLE
COMPAS variable: *derived from* BaseBinaryStar::m_SemiMajorAxis
Description: Semi-major axis at ZAMS ($R_\odot$).
Header String: Separation@ZAMS

Note that this property has the same header string as SEMI_MAJOR_AXIS. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SIMULTANEOUS_RLOF

Data type: BOOL
COMPAS variable: BaseBinaryStar::m_RLOFDetails.simultaneousRLOF
Description: Flag to indicate that both stars are undergoing RLOF.
Header String: Simultaneous_RLOF

STABLE_RLOF_POST_COMMON_ENVELOPE

Data type: BOOL
COMPAS variable: BaseBinaryStar::m_RLOFDetails.stableRLOFPostCEE
Description: Flag to indicate stable mass transfer after common envelope event.
Header String: Stable_RLOF>CE

STELLAR_MERGER

Data type: BOOL
COMPAS variable: BaseBinaryStar::m_StellarMerger
Description: Flag to indicate the stars merged (were touching) during evolution.
Header String: Merger

STELLAR_MERGER_AT_BIRTH

Data type: BOOL
COMPAS variable: BaseBinaryStar::m_StellarMergerAtBirth
Description: Flag to indicate the stars merged (were touching) at birth.
Header String: Merger_At_Birth

STELLAR_TYPE_1_POST_COMMON_ENVELOPE

Data type: INT
COMPAS variable: BinaryConstituentStar::m_CEDetails.postCEE.stellarType
Description: Stellar type (per Hurley et al. (2000)) of the primary star immediately following common envelope event.
Header String: Stellar_Type_1>CE

Note that this property has the same header string as STELLAR_TYPE_NAME_1_POST_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_1_PRE_COMMON_ENVELOPE

Data type:	INT
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.stellarType
Description:	Stellar type (per Hurley et al. (2000)) of the primary star at the onset of RLOF leading to the common-envelope episode.
Header String:	Stellar_Type_1<CE

Note that this property has the same header string as STELLAR_TYPE_NAME_1_PRE_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_2_POST_COMMON_ENVELOPE

Data type:	INT
COMPAS variable:	BinaryConstituentStar::m_CEDetails.postCEE.stellarType
Description:	Stellar type (per Hurley et al. (2000)) of the secondary star immediately following common envelope event.
Header String:	Stellar_Type_2>CE

Note that this property has the same header string as STELLAR_TYPE_NAME_2_POST_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_2_PRE_COMMON_ENVELOPE

Data type:	INT
COMPAS variable:	BinaryConstituentStar::m_CEDetails.preCEE.stellarType
Description:	Stellar type (per Hurley et al. (2000)) of the secondary star at the onset of RLOF leading to the common-envelope episode.
Header String:	Stellar_Type_2<CE

Note that this property has the same header string as STELLAR_TYPE_NAME_2_PRE_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_NAME_1_POST_COMMON_ENVELOPE

Data type:	STRING
COMPAS variable:	*derived from* BinaryConstituentStar::m_CEDetails.postCEE.stellarType
Description:	Stellar type name (per Hurley et al. (2000)) of the primary star immediately following common envelope event. e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header String:	Stellar_Type_1>CE

Note that this property has the same header string as STELLAR_TYPE_1_POST_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_NAME_1_PRE_COMMON_ENVELOPE

Data type: STRING
COMPAS variable: *derived from* BinaryConstituentStar::m_CEDetails.preCEE.stellarType
Description: Stellar type name (per Hurley et al. (2000)) of the primary star at the onset of RLOF leading to the common-envelope episode.
e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header String: Stellar_Type_1<CE

Note that this property has the same header string as STELLAR_TYPE_1_PRE_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_NAME_2_POST_COMMON_ENVELOPE

Data type: STRING
COMPAS variable: *derived from* BinaryConstituentStar::m_CEDetails.postCEE.stellarType
Description: Stellar type name (per Hurley et al. (2000)) of the secondary star immediately following common envelope event.
e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header String: Stellar_Type_2>CE

Note that this property has the same header string as STELLAR_TYPE_2_POST_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

STELLAR_TYPE_NAME_2_PRE_COMMON_ENVELOPE

Data type: STRING
COMPAS variable: *derived from* BinaryConstituentStar::m_CEDetails.preCEE.stellarType
Description: Stellar type name (per Hurley et al. (2000)) of the secondary star at the onset of RLOF leading to the common-envelope episode.
e.g. "First_Giant_Branch", "Core_Helium_Burning", "Helium_White_Dwarf", etc.
Header String: Stellar_Type_2<CE

Note that this property has the same header string as STELLAR_TYPE_2_PRE_COMMON_ENVELOPE. *It is expected that one or the other is printed in any file, but not both. If both are printed then the file will contain two columns with the same header string.*

SUPERNOVA_STATE

Data type: INT
COMPAS variable: *derived from* BaseBinaryStar::m_SupernovaState
Description: Indicates which star(s) went supernova. Will be printed as one of:

No supernova	=	0
Star 1 is the supernova	=	1
Star 2 is the supernova	=	2
Both stars are supernovae	=	3

Header String: Supernova_State

SYNCHRONIZATION_TIMESCALE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_SynchronizationTimescale
Description:	Tidal synchronisation timescale (Myr).
Header String:	Tau_Sync

SYSTEMIC_SPEED

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_SystemicVelocity
Description:	Post-supernova systemic (centre-of-mass) velocity (km s^{-1}).
Header String:	Systemic_Velocity

TIME

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_Time
Description:	Time since ZAMS (Myr).
Header String:	Time

TIME_TO_COALESCENCE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_TimeToCoalescence
Description:	Time between formation of double compact object and gravitational-wave driven merger (Myr).
Header String:	Coalescence_Time

TOTAL_ANGULAR_MOMENTUM

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_TotalAngularMomentum
Description:	Total angular momentum calculated using regular conservation of energy (Msol AU^2 yr^{-1}).
Header String:	Ang_Momentum_Total

TOTAL_ENERGY

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_TotalEnergy
Description:	Total energy calculated using regular conservation of energy (Msol AU^2 yr^{-1}).
Header String:	Energy_Total

UNBOUND

Data type:	BOOL
COMPAS variable:	BaseBinaryStar::m_Unbound
Description:	Flag to indicate the binary is unbound (or has become unbound after a supernova event).
Header String:	Unbound

ZETA_LOBE

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_ZetaLobe
Description:	The logarithmic derivative of Roche lobe radius with respect to donor mass for q= $\frac{\text{Md}}{\text{Ma}}$ at the onset of the RLOF.
Header String:	Zeta_Lobe

ZETA_STAR

Data type:	DOUBLE
COMPAS variable:	BaseBinaryStar::m_ZetaStar
Description:	Mass-radius exponent of the star at the onset of the RLOF. Calculated differently based on the value of program option *--zeta-prescription*
Header String:	Zeta_Star

Log File Record Specification: Program Options

As described in Standard Log File Record Specifiers, when specifying known properties in a log file record specification record, the property name must be prefixed with the property type.

Currently there is a single program option property type available for use: PROGRAM_OPTION.

For example, to specify the program option property RANDOM_SEED, use:

PROGRAM_OPTION::RANDOM_SEED

Following is the list of program option properties available for inclusion in log file record specifiers.

Program Option Properties

ADD_OPTIONS_TO_SYSPARMS

Data type:	INT
COMPAS variable:	Options::m_AddOptionsToSysParms
Description:	Value of program option --*add-options-to-sysparms*
Header String:	Add_Options_To_SysParms

ALLOW_MS_STAR_TO_SURVIVE_COMMON_ENVELOPE

Data type:	BOOL
COMPAS variable:	Options::m_AllowMainSequenceStarToSurviveCommonEnvelope
Description:	Value of program option --*common-envelope-allow-main-sequence-survive*
Header String:	Allow_MS_To_Survive_CE

ALLOW_RLOF_AT_BIRTH

Data type:	BOOL
COMPAS variable:	Options::m_AllowRLOFAtBirth
Description:	Value of program option --*allow-rlof-at-birth*
Header String:	Allow_RLOF@Birth

ALLOW_TOUCHING_AT_BIRTH

Data type:	BOOL
COMPAS variable:	Options::m_AllowTouchingAtBirth
Description:	Value of program option --*allow-touching-at-birth*
Header String:	Allow_Touching@Birth

ANG_MOM_CONSERVATION_DURING_CIRCULARISATION

Data type:	BOOL
COMPAS variable:	Options::m_AngularMomentumConservationDuringCircularisation
Description:	Value of program option --*angular-momentum-conservation-during-circularisation*
Header String:	Conserve_AngMom@Circ

BLACK_HOLE_KICKS

Data type:	INT
COMPAS variable:	Options::m_BlackHoleKicks
Description:	Value of program option --*black-hole-kicks*
Header String:	BH_Kicks

CASE_BB_STABILITY_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_CaseBBStabilityPrescription
Description:	Value of program option --*case-BB-stability-prescription*
Header String:	BB_Mass_xFer_Stblty_Prscrptn

CHE_MODE

Data type:	INT
COMPAS variable:	Options::m_CheMode
Description:	Value of program option --*chemically-homogeneous-evolution*
Header String:	CHE_Mode

CIRCULARISE_BINARY_DURING_MT

Data type:	BOOL
COMPAS variable:	Options::m_CirculariseBinaryDuringMassTransfer
Description:	Value of program option --*circularise-binary-during-mass-transfer*
Header String:	Circularise@MT

COMMON_ENVELOPE_ALPHA

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeAlpha
Description:	Value of program option --*common-envelope-alpha*
Header String:	CE_Alpha

COMMON_ENVELOPE_ALPHA_THERMAL

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeAlphaThermal
Description:	Value of program option --*common-envelope-alpha-thermal*
Header String:	CE_Alpha_Thermal

COMMON_ENVELOPE_LAMBDA

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeLambda
Description:	Value of program option --*common-envelope-lambda*
Header String:	CE_Lambda

COMMON_ENVELOPE_LAMBDA_MULTIPLIER

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeLambdaMultiplier
Description:	Value of program option --*common-envelope-lambda-multiplier*
Header String:	CE_Lambda_Multiplier

COMMON_ENVELOPE_LAMBDA_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_CommonEnvelopeLambdaPrescription
Description:	Value of program option --*common-envelope-lambda-prescription*
Header String:	CE_Lambda_Prscrptn

COMMON_ENVELOPE_MASS_ACCRETION_CONSTANT

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeMassAccretionConstant
Description:	Value of program option --*common-envelope-mass-accretion-constant*
Header String:	CE_Mass_Accr_Constant

COMMON_ENVELOPE_MASS_ACCRETION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeMassAccretionMax
Description:	Value of program option --*common-envelope-mass-accretion-max*
Header String:	CE_Mass_Accr_Max

COMMON_ENVELOPE_MASS_ACCRETION_MIN

Data type:	DOUBLE
COMPAS variable:	Options::m_CommonEnvelopeMassAccretionMin
Description:	Value of program option --*common-envelope-mass-accretion-min*
Header String:	CE_Mass_Accr_Min

COMMON_ENVELOPE_MASS_ACCRETION_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_CommonEnvelopeMassAccretionPrescription
Description:	Value of program option --*common-envelope-mass-accretion-prescription*
Header String:	CE_Mass_Accr_Prscrptn

COMMON_ENVELOPE_RECOMBINATION_ENERGY_DENSITY

Data type: DOUBLE
COMPAS variable: Options::m_CommonEnvelopeRecombinationEnergyDensity
Description: Value of program option --*common-envelope-recombination-energy-density*
Header String: CE_Recomb_Enrgy_Dnsty

COMMON_ENVELOPE_SLOPE_KRUCKOW

Data type: DOUBLE
COMPAS variable: Options::m_CommonEnvelopeSlopeKruckow
Description: Value of program option --*common-envelope-slope-kruckow*
Header String: CE_Slope_Kruckow

ECCENTRICITY

Data type: DOUBLE
COMPAS variable: Options::m_Eccentricity
Description: Value of program option --*eccentricity*
Header String: Eccentricity

ECCENTRICITY_DISTRIBUTION

Data type: INT
COMPAS variable: Options::m_EccentricityDistribution
Description: Value of program option --*eccentricity-distribution*
Header String: Eccentricity_Dstrbtn

ECCENTRICITY_DISTRIBUTION_MAX

Data type: DOUBLE
COMPAS variable: Options::m_EccentricityDistributionMax
Description: Value of program option --*eccentricity-max*
Header String: Eccentricity_Dstrbtn_Max

ECCENTRICITY_DISTRIBUTION_MIN

Data type: DOUBLE
COMPAS variable: Options::m_EccentricityDistributionMin
Description: Value of program option --*eccentricity-min*
Header String: Eccentricity_Dstrbtn_Min

EDDINGTON_ACCRETION_FACTOR

Data type: DOUBLE
COMPAS variable: Options::m_EddingtonAccretionFactor
Description: Value of program option --*eddington-accretion-factor*
Header String: Eddington_Accr_Factor

ENVELOPE_STATE_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_EnvelopeStatePrescription
Description:	Value of program option --*envelope-state-prescription*
Header String:	Envelope_State_Prscrptn

FRYER_SUPERNOVA_ENGINE

Data type:	INT
COMPAS variable:	Options::m_FryerSupernovaEngine
Description:	Value of program option --*fryer-supernova-engine*
Header String:	Fryer_SN_Engine

INITIAL_MASS

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMass
Description:	Value of program option --*initial-mass*
Header String:	Initial_Mass

INITIAL_MASS_1

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMass1
Description:	Value of program option --*initial-mass-1*
Header String:	Initial_Mass_1

INITIAL_MASS_2

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMass2
Description:	Value of program option --*initial-mass-2*
Header String:	Initial_Mass_2

INITIAL_MASS_FUNCTION

Data type:	INT
COMPAS variable:	Options::m_InitialMassFunction
Description:	Value of program option --*initial-mass-function*
Header String:	Initial_Mass_Function

INITIAL_MASS_FUNCTION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMassFunctionMax
Description:	Value of program option --*initial-mass-max*
Header String:	Initial_Mass_Func_Max

INITIAL_MASS_FUNCTION_MIN

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMassFunctionMin
Description:	Value of program option --*initial-mass-min*
Header String:	Initial_Mass_Func_Min

INITIAL_MASS_FUNCTION_POWER

Data type:	DOUBLE
COMPAS variable:	Options::m_InitialMassFunctionPower
Description:	Value of program option --*initial-mass-power*
Header String:	Initial_Mass_Func_Power

KICK_DIRECTION_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_KickDirectionDistribution
Description:	Value of program option --*kick-direction*
Header String:	Kick_Direction_Dstrbtn

KICK_DIRECTION_POWER

Data type:	DOUBLE
COMPAS variable:	Options::m_KickDirectionPower
Description:	Value of program option --*kick-direction-power*
Header String:	Kick_Direction_Power

KICK_SCALING_FACTOR

Data type:	DOUBLE
COMPAS variable:	Options::m_KickScalingFactor
Description:	Value of program option --*kick-scaling-factor*
Header String:	Kick_Scaling_Factor

KICK_MAGNITUDE

Data type:	DOUBLE
COMPAS variable:	Options::m_KickMagnitude
Description:	Value of program option --*kick-magnitude*
Header String:	Kick_Magnitude

KICK_MAGNITUDE_1

Data type:	DOUBLE
COMPAS variable:	Options::m_KickMagnitude1
Description:	Value of program option --*kick-magnitude-1*
Header String:	Kick_Magnitude_1

KICK_MAGNITUDE_2

Data type: DOUBLE
COMPAS variable: Options::m_KickMagnitude2
Description: Value of program option --*kick-magnitude-2*
Header String: Kick_Magnitude_2

KICK_MAGNITUDE_DISTRIBUTION

Data type: INT
COMPAS variable: Options::m_KickMagnitudeDistribution
Description: Value of program option --*kick-magnitude-distribution*
Header String: Kick_Magnitude_Dstrbtn

KICK_MAGNITUDE_DISTRIBUTION_MAXIMUM

Data type: DOUBLE
COMPAS variable: Options::m_KickMagnitudeDistributionMaximum
Description: Value of program option --*kick-magnitude-max*
Header String: Kick_Magnitude_Dstrbtn_Max

KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_BH

Data type: DOUBLE
COMPAS variable: Options::kickMagnitudeDistributionSigmaCCSN_BH
Description: Value of program option --*kick-magnitude-sigma-CCSN-BH*
Header String: Sigma_Kick_CCSN_BH

KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_NS

Data type: DOUBLE
COMPAS variable: Options::kickMagnitudeDistributionSigmaCCSN_NS
Description: Value of program option --*kick-magnitude-sigma-CCSN-NS*
Header String: Sigma_Kick_CCSN_NS

KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_ECSN

Data type: DOUBLE
COMPAS variable: Options::kickMagnitudeDistributionSigmaForECSN
Description: Value of program option --*kick-magnitude-sigma-ECSN*
Header String: Sigma_Kick_ECSN

KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_USSN

Data type: DOUBLE
COMPAS variable: Options::kickMagnitudeDistributionSigmaForUSSN
Description: Value of program option --*kick-magnitude-sigma-USSN*
Header String: Sigma_Kick_USSN

KICK_MEAN_ANOMALY_1

Data type: DOUBLE
COMPAS variable: Options::m_KickMeanAnomaly1
Description: Value of program option --*kick-mean-anomaly-1*
Header String: Kick_Mean_Anomaly_1

KICK_MEAN_ANOMALY_2

Data type: DOUBLE
COMPAS variable: Options::m_KickMeanAnomaly2
Description: Value of program option --*kick-mean-anomaly-2*
Header String: Kick_Mean_Anomaly_2

KICK_MAGNITUDE_RANDOM

Data type: DOUBLE
COMPAS variable: Options::m_KickMagnitudeRandom
Description: Value of program option --*kick-magnitude-random*
Header String: Kick_Magnitude_Random

KICK_MAGNITUDE_RANDOM_1

Data type: DOUBLE
COMPAS variable: Options::m_KickMagnitudeRandom1
Description: Value of program option --*kick-magnitude-random-1*
Header String: Kick_Magnitude_Random_1

KICK_MAGNITUDE_RANDOM_2

Data type: DOUBLE
COMPAS variable: Options::m_KickMagnitudeRandom2
Description: Value of program option --*kick-magnitude-random-2*
Header String: Kick_Magnitude_Random_2

KICK_PHI_1

Data type: DOUBLE
COMPAS variable: Options::m_KickPhi1
Description: Value of program option --*kick-phi-1*
Header String: Kick_Mean_Phi_1

KICK_PHI_2

Data type: DOUBLE
COMPAS variable: Options::m_KickPhi2
Description: Value of program option --*kick-phi-2*
Header String: Kick_Mean_Phi_2

KICK_THETA_1

Data type:	DOUBLE
COMPAS variable:	Options::m_KickTheta1
Description:	Value of program option --*kick-theta-1*
Header String:	Kick_Mean_Theta_1

KICK_THETA_2

Data type:	DOUBLE
COMPAS variable:	Options::m_KickTheta2
Description:	Value of program option --*kick-theta-2*
Header String:	Kick_Mean_Theta_2

LBV_FACTOR

Data type:	DOUBLE
COMPAS variable:	Options::m_LuminousBlueVariableFactor
Description:	Value of program option --*luminous-blue-variable-multiplier*
Header String:	LBV_Factor

LBV_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_LuminousBlueVariablePrescription
Description:	Value of program option --*luminous-blue-variable-prescription*
Header String:	LBV_Mass_Loss_Prscrptn

MASS_LOSS_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_MassLossPrescription
Description:	Value of program option --*mass-loss-prescription*
Header String:	Mass_Loss_Prscrptn

MASS_RATIO

Data type:	DOUBLE
COMPAS variable:	Options::m_MassRatio
Description:	Value of program option --*mass-ratio*
Header String:	Mass_Ratio

MASS_RATIO_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_MassRatioDistribution
Description:	Value of program option --*mass-ratio-distribution*
Header String:	Mass_Ratio_Dstrbtn

MASS_RATIO_DISTRIBUTION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_MassRatioDistributionMax
Description:	Value of program option −−*mass-ratio-max*
Header String:	Mass_Ratio_Dstrbtn_Max

MASS_RATIO_DISTRIBUTION_MIN

Data type:	DOUBLE
COMPAS variable:	Options::m_MassRatioDistributionMin
Description:	Value of program option −−*mass-ratio-min*
Header String:	Mass_Ratio_Dstrbtn_Min

MAXIMUM_EVOLUTION_TIME

Data type:	DOUBLE
COMPAS variable:	Options::m_MaxEvolutionTime
Description:	Value of program option −−*maximum-evolution-time*
Header String:	Max_Evolution_Time

MAXIMUM_DONOR_MASS

Data type:	DOUBLE
COMPAS variable:	Options::m_MaximumMassDonorNandezIvanova
Description:	Value of program option −−*maximum-mass-donor-nandez-ivanova*
Header String:	Max_Donor_Mass

MAXIMUM_NEUTRON_STAR_MASS

Data type:	DOUBLE
COMPAS variable:	Options::m_MaximumNeutronStarMass
Description:	Value of program option −−*maximum-neutron-star-mass*
Header String:	Max_NS_Mass

MAXIMUM_TIMESTEPS

Data type:	INT
COMPAS variable:	Options::m_MaxNumberOfTimestepIterations
Description:	Value of program option −−*maximum-number-timestep-iterations*
Header String:	Max_Timesteps

MCBUR1

Data type:	DOUBLE
COMPAS variable:	Options::m_mCBUR1
Description:	Value of program option −−*mcbur1*
Header String:	MCBUR1

METALLICITY

Data type: DOUBLE
COMPAS variable: Options::m_Metallicity
Description: Value of program option *--metallicity*
Header String: Metallicity

METALLICITY_DISTRIBUTION

Data type: INT
COMPAS variable: Options::m_MetallicityDistribution
Description: Value of program option *--metallicity-distribution*
Header String: Metallicity_Dstrbtn

METALLICITY_DISTRIBUTION_MAX

Data type: DOUBLE
COMPAS variable: Options::m_MetallicityDistributionMax
Description: Value of program option *--metallicity-max*
Header String: Metallicity_Dstrbtn_Max

METALLICITY_DISTRIBUTION_MIN

Data type: DOUBLE
COMPAS variable: Options::m_MetallicityDistributionMin
Description: Value of program option *--metallicity-min*
Header String: Metallicity_Dstrbtn_Min

MINIMUM_MASS_SECONDARY

Data type: DOUBLE
COMPAS variable: Options::m_MinimumMassSecondary
Description: Value of program option *--minimum-secondary-mass*
Header String: Min_Secondary_Mass

MT_ACCRETION_EFFICIENCY_PRESCRIPTION

Data type: INT
COMPAS variable: Options::m_MassTransferAccretionEfficiencyPrescription
Description: Value of program option *--mass-transfer-accretion-efficiency-prescription*
Header String: MT_Acc_Efficiency_Prscrptn

MT_ANG_MOM_LOSS_PRESCRIPTION

Data type: INT
COMPAS variable: Options::m_MassTransferAngularMomentumLossPrescription
Description: Value of program option *--mass-transfer-angular-momentum-loss-prescription*
Header String: MT_AngMom_Loss_Prscrptn

MT_FRACTION_ACCRETED

Data type:	DOUBLE
COMPAS variable:	Options::m_MassTransferFractionAccreted
Description:	Value of program option --*mass-transfer-fa*
Header String:	MT_Fraction_Accreted

MT_JLOSS

Data type:	DOUBLE
COMPAS variable:	Options::m_MassTransferJloss
Description:	Value of program option --*mass-transfer-jloss*
Header String:	MT_JLoss

MT_THERMAL_LIMIT_C

Data type:	DOUBLE
COMPAS variable:	Options::m_MassTransferCParameter
Description:	Value of program option --*mass-transfer-thermal-limit-C*
Header String:	MT_Thermal_Limit_C

MT_REJUVENATION_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_MassTransferRejuvenationPrescription
Description:	Value of program option --*mass-transfer-rejuvenation-prescription*
Header String:	MT_Rejuvenation_Prscrptn

MT_THERMALLY_LIMITED_VARIATION

Data type:	INT
COMPAS variable:	Options::m_MassTransferThermallyLimitedVariation
Description:	Value of program option --*mass-transfer-thermal-limit-accretor*
Header String:	MT_Thermally_Lmtd_Variation

MULLER_MANDEL_KICK_MULTIPLIER_BH

Data type:	DOUBLE
COMPAS variable:	Options::m_MullerMandelKickBH
Description:	Value of program option --*muller-mandel-kick-multiplier-BH*
Header String:	MM_Kick_Multiplier_BH

MULLER_MANDEL_KICK_MULTIPLIER_NS

Data type:	DOUBLE
COMPAS variable:	Options::m_MullerMandelKickNS
Description:	Value of program option --*muller-mandel-kick-multiplier-NS*
Header String:	MM_Kick_Multiplier_NS

NEUTRINO_MASS_LOSS_ASSUMPTION_BH

Data type:	INT
COMPAS variable:	Options::m_NeutrinoMassLossAssumptionBH
Description:	Value of program option *--neutrino-mass-loss-BH-formation*
Header String:	Neutrino_Mass_Loss_Assmptn

NEUTRINO_MASS_LOSS_VALUE_BH

Data type:	DOUBLE
COMPAS variable:	Options::m_NeutrinoMassLossValueBH
Description:	Value of program option *--neutrino-mass-loss-BH-formation-value*
Header String:	Neutrino_Mass_Loss_Value

NS_EOS

Data type:	INT
COMPAS variable:	Options::m_NeutronStarEquationOfState
Description:	Value of program option *--neutron-star-equation-of-state*
Header String:	NS_EOS

ORBITAL_PERIOD

Data type:	DOUBLE
COMPAS variable:	Options::m_OrbitalPeriod
Description:	Value of program option *--orbital-period*
Header String:	Orbital_Period

ORBITAL_PERIOD_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_OrbitalPeriodDistribution
Description:	Value of program option *--orbital-period-distribution*
Header String:	Orbital_Period_Dstrbtn

ORBITAL_PERIOD_DISTRIBUTION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_OrbitalPeriodDistributionMax
Description:	Value of program option *--orbital-period-max*
Header String:	Orbital_Period_Max

ORBITAL_PERIOD_DISTRIBUTION_MIN

Data type:	DOUBLE
COMPAS variable:	Options::m_OrbitalPeriodDistributionMin
Description:	Value of program option *--orbital-period-min*
Header String:	Orbital_Period_Min

PISN_LOWER_LIMIT

Data type:	DOUBLE
COMPAS variable:	Options::m_PairInstabilityLowerLimit
Description:	Value of program option *--PISN-lower-limit*
Header String:	PISN_Lower_Limit

PISN_UPPER_LIMIT

Data type:	DOUBLE
COMPAS variable:	Options::m_PairInstabilityUpperLimit
Description:	Value of program option *--PISN-upper-limit*
Header String:	PISN_Upper_Limit

PPI_LOWER_LIMIT

Data type:	DOUBLE
COMPAS variable:	Options::m_PulsationalPairInstabilityLowerLimit
Description:	Value of program option *--PPI-lower-limit*
Header String:	PPI_Lower_Limit

PPI_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_PulsationalPairInstabilityPrescription
Description:	Value of program option *--pulsational-pair-instability-prescription*
Header String:	PPI_Prscrptn

PPI_UPPER_LIMIT

Data type:	DOUBLE
COMPAS variable:	Options::m_PulsationalPairInstabilityUpperLimit
Description:	Value of program option *--PPI-upper-limit*
Header String:	PPI_Upper_Limit

PULSAR_MAGNETIC_FIELD_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_PulsarBirthMagneticFieldDistribution
Description:	Value of program option *--pulsar-birth-magnetic-field-distribution*
Header String:	Pulsar_Mag_Field_Dstrbtn

PULSAR_MAGNETIC_FIELD_DISTRIBUTION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_PulsarBirthMagneticFieldDistributionMax
Description:	Value of program option *--pulsar-birth-magnetic-field-distribution-max*
Header String:	Pulsar_Mag_Field_Dstrbtn_Max

PULSAR_MAGNETIC_FIELD_DISTRIBUTION_MIN

Data type: DOUBLE
COMPAS variable: Options::m_PulsarBirthMagneticFieldDistributionMin
Description: Value of program option *--pulsar-birth-magnetic-field-distribution-min*
Header String: Pulsar_Mag_Field_Dstrbtn_Min

PULSAR_BIRTH_SPIN_PERIOD_DISTRIBUTION

Data type: INT
COMPAS variable: Options::m_PulsarBirthSpinPeriodDistribution
Description: Value of program option *--pulsar-birth-spin-period-distribution*
Header String: Pulsar_Spin_Period_Dstrbtn

PULSAR_BIRTH_SPIN_DISTRIBUTION_MAX

Data type: DOUBLE
COMPAS variable: Options::m_PulsarBirthSpinPeriodDistributionMax
Description: Value of program option *--pulsar-birth-spin-period-distribution-max*
Header String: Pulsar_Spin_Period_Dstrbtn_Max

PULSAR_BIRTH_SPIN_DISTRIBUTION_MIN

Data type: DOUBLE
COMPAS variable: Options::m_PulsarBirthSpinPeriodDistributionMin
Description: Value of program option *--pulsar-birth-spin-period-distribution-min*
Header String: Pulsar_Spin_Period_Dstrbtn_Min

PULSAR_MAGNETIC_FIELD_DECAY_MASS_SCALE

Data type: DOUBLE
COMPAS variable: Options:m_PulsarMagneticFieldDecayMassscale
Description: Value of program option *--pulsar-magnetic-field-decay-massscale*
Header String: Pulsar_Mag_Field_Decay_mScale

PULSAR_MAGNETIC_FIELD_DECAY_TIME_SCALE

Data type: DOUBLE
COMPAS variable: Options:m_PulsarMagneticFieldDecayTimescale
Description: Value of program option *--pulsar-magnetic-field-decay-timescale*
Header String: Pulsar_Mag_Field_Decay_tScale

PULSAR_MINIMUM_MAGNETIC_FIELD

Data type: DOUBLE
COMPAS variable: Options::m_PulsarLog10MinimumMagneticField
Description: Value of program option *--pulsar-minimum-magnetic-field*
Header String: Pulsar_Minimum_Mag_Field

RANDOM_SEED

Data type:	UNSIGNED LONG INT
COMPAS variable:	Options::randomSeed
Description:	Value of program option --*random-seed*
Header String:	SEED(OPTION)

RANDOM_SEED_CMDLINE

Data type:	UNSIGNED LONG INT
COMPAS variable:	Options::randomSeed
Description:	Value of program option --*random-seed* (specified on the commandline)
Header String:	SEED(CMDLINE)

REMNANT_MASS_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_RemnantMassPrescription
Description:	Value of program option --*remnant-mass-prescription*
Header String:	Remnant_Mass_Prscrptn

ROTATIONAL_VELOCITY_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_RotationalVelocityDistribution
Description:	Value of program option --*rotational-velocity-distribution*
Header String:	Rotational_Velocity_Dstrbtn

SEMI_MAJOR_AXIS

Data type:	DOUBLE
COMPAS variable:	Options::m_SemiMajorAxis
Description:	Value of program option --*semi-major-axis*
Header String:	Semi-Major_Axis

SEMI_MAJOR_AXIS_DISTRIBUTION

Data type:	INT
COMPAS variable:	Options::m_SemiMajorAxisDistribution
Description:	Value of program option --*semi-major-axis-distribution*
Header String:	Semi-Major_Axis_Dstrbtn

SEMI_MAJOR_AXIS_DISTRIBUTION_MAX

Data type:	DOUBLE
COMPAS variable:	Options::m_SemiMajorAxisDistributionMax
Description:	Value of program option --*semi-major-axis-max*
Header String:	Semi-Major_Axis_Dstrbtn_Max

SEMI_MAJOR_AXIS_DISTRIBUTION_MIN

Data type:	DOUBLE
COMPAS variable:	Options::m_SemiMajorAxisDistributionMin
Description:	Value of program option *--semi-major-axis-min*
Header String:	Semi-Major_Axis_Dstrbtn_Min

STELLAR_ZETA_PRESCRIPTION

Data type:	INT
COMPAS variable:	Options::m_StellarZetaPrescription
Description:	Value of program option *--stellar-zeta-prescription*
Header String:	Stellar_Zeta_Prscrptn

WR_FACTOR

Data type:	DOUBLE
COMPAS variable:	Options::m_WolfRayetFactor
Description:	Value of program option *--wolf-rayet-multiplier*
Header String:	WR_Factor

ZETA_ADIABATIC_ARBITRARY

Data type:	DOUBLE
COMPAS variable:	Options::m_ZetaAdiabaticArbitrary
Description:	Value of program option *--zeta-adiabatic-arbitrary*
Header String:	Zeta_Adiabatic_Arbitrary

ZETA_MS

Data type:	DOUBLE
COMPAS variable:	Options::m_ZetaMainSequence
Description:	Value of program option *--zeta-main-sequence*
Header String:	Zeta_Main_Sequence_Giant

ZETA_RADIATIVE_ENVELOPE_GIANT

Data type:	DOUBLE
COMPAS variable:	Options::m_ZetaRadiativeEnvelopeGiant
Description:	Value of program option *--zeta-radiative-envelope-giant*
Header String:	Zeta_Radiative_Envelope_Giant

Default Log File Record Specifications

Following are the default log file record specifications for each of the standard log files. These specifications can be overridden by the use of a log file specifications file via the logfile-definitions program option.

SSE System Parameters

```
const ANY_PROPERTY_VECTOR SSE_SYSTEM_PARAMETERS_REC = {
    STAR_PROPERTY::RANDOM_SEED,
    STAR_PROPERTY::MZAMS,
    STAR_PROPERTY::RZAMS,
    STAR_PROPERTY::METALLICITY,
    STAR_PROPERTY::OMEGA_ZAMS,
    STAR_PROPERTY::INITIAL_STELLAR_TYPE,
    STAR_PROPERTY::STELLAR_TYPE,
    STAR_PROPERTY::SUPERNOVA_KICK_MAGNITUDE_RANDOM_NUMBER,
    STAR_PROPERTY::MASS,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_NS,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_BH,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_ECSN,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_USSN,
    PROGRAM_OPTION::LBV_FACTOR,
    PROGRAM_OPTION::WR_FACTOR
};
```

SSE Detailed Output

```
const ANY_PROPERTY_VECTOR SSE_DETAILED_OUTPUT_REC = {
    STAR_PROPERTY::AGE,
    STAR_PROPERTY::DT,
    STAR_PROPERTY::TIME,
    STAR_PROPERTY::STELLAR_TYPE,
    STAR_PROPERTY::METALLICITY,
    STAR_PROPERTY::MASS_0,
    STAR_PROPERTY::MASS,
    STAR_PROPERTY::RADIUS,
    STAR_PROPERTY::RZAMS,
    STAR_PROPERTY::LUMINOSITY,
    STAR_PROPERTY::TEMPERATURE,
    STAR_PROPERTY::CORE_MASS,
    STAR_PROPERTY::CO_CORE_MASS,
    STAR_PROPERTY::HE_CORE_MASS,
    STAR_PROPERTY::MDOT,
    STAR_PROPERTY::DOMINANT_MASS_LOSS_RATE,
    STAR_PROPERTY::TIMESCALE_MS
};
```

SSE Supernovae

```
const ANY_PROPERTY_VECTOR SSE_SUPERNOVA_REC = {
    STAR_PROPERTY::RANDOM_SEED,
    STAR_PROPERTY::DRAWN_KICK_MAGNITUDE,
    STAR_PROPERTY::KICK_MAGNITUDE,
    STAR_PROPERTY::FALLBACK_FRACTION,
    STAR_PROPERTY::TRUE_ANOMALY,
    STAR_PROPERTY::SUPERNOVA_THETA,
    STAR_PROPERTY::SUPERNOVA_PHI,
    STAR_PROPERTY::SN_TYPE,
    STAR_PROPERTY::TOTAL_MASS_AT_COMPACT_OBJECT_FORMATION,
    STAR_PROPERTY::CO_CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    STAR_PROPERTY::MASS,
    STAR_PROPERTY::STELLAR_TYPE,
    STAR_PROPERTY::STELLAR_TYPE_PREV,
    STAR_PROPERTY::CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    STAR_PROPERTY::HE_CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    STAR_PROPERTY::TIME,
    STAR_PROPERTY::IS_HYDROGEN_POOR
};
```

SSE Switch Log

const ANY_PROPERTY_VECTOR SSE_SWITCH_LOG_REC = {

STAR_PROPERTY::RANDOM_SEED,
STAR_PROPERTY::TIME

};

The default record specification can be modified at runtime via a logfile record spcifications file (program option *--logfile-definitions*).

Note that the SSE Switch Log file has the following columns automatically appended to each record:

- The stellar type from which the star is switching.
- The stellar type from which the star is switching.

SWITCHING_FROM

Data type:	INT
COMPAS variable:	*derived from* BaseStar::m_StellarType
Description:	The stellar type of the constituent star immediately prior to the switch.
Header String:	SWITCHING_FROM

SWITCHING_TO

Data type:	INT
COMPAS variable:	*Not applicable*
Description:	The stellar type to which the constituent star will switch (i.e. the stellar type immediately following the switch.
Header String:	SWITCHING_TO

These columns will always be automatically appended to each SSE Switch Log record: they cannot be removed via the logfile record specifications file.

BSE System Parameters

```
const ANY_PROPERTY_VECTOR BSE_SYSTEM_PARAMETERS_REC = {
    BINARY_PROPERTY::RANDOM_SEED,
    STAR_1_PROPERTY::MZAMS,
    STAR_2_PROPERTY::MZAMS,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_INITIAL,
    BINARY_PROPERTY::ECCENTRICITY_INITIAL,
    STAR_1_PROPERTY::SUPERNOVA_KICK_MAGNITUDE_RANDOM_NUMBER,
    STAR_1_PROPERTY::SUPERNOVA_THETA,
    STAR_1_PROPERTY::SUPERNOVA_PHI,
    STAR_1_PROPERTY::MEAN_ANOMALY,
    STAR_2_PROPERTY::SUPERNOVA_KICK_MAGNITUDE_RANDOM_NUMBER,
    STAR_2_PROPERTY::SUPERNOVA_THETA,
    STAR_2_PROPERTY::SUPERNOVA_PHI,
    STAR_2_PROPERTY::MEAN_ANOMALY,
    STAR_1_PROPERTY::OMEGA_ZAMS,
    STAR_2_PROPERTY::OMEGA_ZAMS,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_NS,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_CCSN_BH,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_ECSN,
    PROGRAM_OPTION::KICK_MAGNITUDE_DISTRIBUTION_SIGMA_FOR_USSN,
    PROGRAM_OPTION::LBV_FACTOR,
    PROGRAM_OPTION::WR_FACTOR,
    PROGRAM_OPTION::COMMON_ENVELOPE_ALPHA,
    STAR_1_PROPERTY::METALLICITY,
    STAR_2_PROPERTY::METALLICITY,
    BINARY_PROPERTY::UNBOUND,
    BINARY_PROPERTY::STELLAR_MERGER,
    BINARY_PROPERTY::STELLAR_MERGER_AT_BIRTH,
    STAR_1_PROPERTY::INITIAL_STELLAR_TYPE,
    STAR_1_PROPERTY::STELLAR_TYPE,
    STAR_2_PROPERTY::INITIAL_STELLAR_TYPE,
    STAR_2_PROPERTY::STELLAR_TYPE,
    BINARY_PROPERTY::ERROR
};
```

BSE Detailed Output

```
const ANY_PROPERTY_VECTOR BSE_DETAILED_OUTPUT_REC = {
    BINARY_PROPERTY::RANDOM_SEED,
    BINARY_PROPERTY::DT,
    BINARY_PROPERTY::TIME,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_RSOL,
    BINARY_PROPERTY::ECCENTRICITY,
    STAR_1_PROPERTY::MZAMS,
    STAR_2_PROPERTY::MZAMS,
    STAR_1_PROPERTY::MASS_0,
    STAR_2_PROPERTY::MASS_0,
    STAR_1_PROPERTY::MASS,
    STAR_2_PROPERTY::MASS,
    STAR_1_PROPERTY::ENV_MASS,
    STAR_2_PROPERTY::ENV_MASS,
    STAR_1_PROPERTY::CORE_MASS,
    STAR_2_PROPERTY::CORE_MASS,
    STAR_1_PROPERTY::HE_CORE_MASS,
    STAR_2_PROPERTY::HE_CORE_MASS,
    STAR_1_PROPERTY::CO_CORE_MASS,
    STAR_2_PROPERTY::CO_CORE_MASS,
    STAR_1_PROPERTY::RADIUS,
    STAR_2_PROPERTY::RADIUS,
    BINARY_PROPERTY::ROCHE_LOBE_RADIUS_1,
    BINARY_PROPERTY::ROCHE_LOBE_RADIUS_2,
    BINARY_PROPERTY::ROCHE_LOBE_TRACKER_1,
    BINARY_PROPERTY::ROCHE_LOBE_TRACKER_2,
    STAR_1_PROPERTY::OMEGA,
    STAR_2_PROPERTY::OMEGA,
    STAR_1_PROPERTY::OMEGA_BREAK,
    STAR_2_PROPERTY::OMEGA_BREAK,
    STAR_1_PROPERTY::INITIAL_STELLAR_TYPE,
    STAR_2_PROPERTY::INITIAL_STELLAR_TYPE,
    STAR_1_PROPERTY::STELLAR_TYPE,
    STAR_2_PROPERTY::STELLAR_TYPE,
    STAR_1_PROPERTY::AGE,
    STAR_2_PROPERTY::AGE,
    STAR_1_PROPERTY::LUMINOSITY,
    STAR_2_PROPERTY::LUMINOSITY,
    STAR_1_PROPERTY::TEMPERATURE,
    STAR_2_PROPERTY::TEMPERATURE,
    STAR_1_PROPERTY::ANGULAR_MOMENTUM,
    STAR_2_PROPERTY::ANGULAR_MOMENTUM,
    STAR_1_PROPERTY::DYNAMICAL_TIMESCALE,
    STAR_2_PROPERTY::DYNAMICAL_TIMESCALE,
```

```
    STAR_1_PROPERTY::THERMAL_TIMESCALE,
    STAR_2_PROPERTY::THERMAL_TIMESCALE,
    STAR_1_PROPERTY::NUCLEAR_TIMESCALE,
    STAR_2_PROPERTY::NUCLEAR_TIMESCALE,
    STAR_1_PROPERTY::ZETA_SOBERMAN,
    STAR_2_PROPERTY::ZETA_SOBERMAN,
    STAR_1_PROPERTY::ZETA_SOBERMAN_HE,
    STAR_2_PROPERTY::ZETA_SOBERMAN_HE,
    STAR_1_PROPERTY::ZETA_HURLEY,
    STAR_2_PROPERTY::ZETA_HURLEY,
    STAR_1_PROPERTY::ZETA_HURLEY_HE,
    STAR_2_PROPERTY::ZETA_HURLEY_HE,
    STAR_1_PROPERTY::MASS_LOSS_DIFF,
    STAR_2_PROPERTY::MASS_LOSS_DIFF,
    STAR_1_PROPERTY::DOMINANT_MASS_LOSS_RATE,
    STAR_2_PROPERTY::DOMINANT_MASS_LOSS_RATE,
    STAR_1_PROPERTY::MASS_TRANSFER_DIFF,
    STAR_2_PROPERTY::MASS_TRANSFER_DIFF,
    BINARY_PROPERTY::TOTAL_ANGULAR_MOMENTUM,
    BINARY_PROPERTY::TOTAL_ENERGY,
    STAR_1_PROPERTY::METALLICITY,
    STAR_2_PROPERTY::METALLICITY,
    BINARY_PROPERTY::MASS_TRANSFER_TRACKER_HISTORY,
    STAR_1_PROPERTY::PULSAR_MAGNETIC_FIELD,
    STAR_2_PROPERTY::PULSAR_MAGNETIC_FIELD,
    STAR_1_PROPERTY::PULSAR_SPIN_FREQUENCY,
    STAR_2_PROPERTY::PULSAR_SPIN_FREQUENCY,
    STAR_1_PROPERTY::PULSAR_SPIN_DOWN_RATE,
    STAR_2_PROPERTY::PULSAR_SPIN_DOWN_RATE,
    STAR_1_PROPERTY::RADIAL_EXPANSION_TIMESCALE,
    STAR_2_PROPERTY::RADIAL_EXPANSION_TIMESCALE
};
```

BSE Double Compact Objects

```
const ANY_PROPERTY_VECTOR BSE_DOUBLE_COMPACT_OBJECTS_REC = {
    BINARY_PROPERTY::RANDOM_SEED,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_AT_DCO_FORMATION,
    BINARY_PROPERTY::ECCENTRICITY_AT_DCO_FORMATION,
    STAR_1_PROPERTY::MASS,
    STAR_1_PROPERTY::STELLAR_TYPE,
    STAR_2_PROPERTY::MASS,
    STAR_2_PROPERTY::STELLAR_TYPE,
    BINARY_PROPERTY::TIME_TO_COALESCENCE,
    BINARY_PROPERTY::TIME,
    BINARY_PROPERTY::MERGES_IN_HUBBLE_TIME,
    STAR_1_PROPERTY::RECYCLED_NEUTRON_STAR,
    STAR_2_PROPERTY::RECYCLED_NEUTRON_STAR
};
```

BSE Common Envelopes

const ANY_PROPERTY_VECTOR BSE_COMMON_ENVELOPES_REC = {

BINARY_PROPERTY::RANDOM_SEED,
BINARY_PROPERTY::TIME,
STAR_1_PROPERTY::LAMBDA_AT_COMMON_ENVELOPE,
STAR_2_PROPERTY::LAMBDA_AT_COMMON_ENVELOPE,
STAR_1_PROPERTY::BINDING_ENERGY_PRE_COMMON_ENVELOPE,
STAR_2_PROPERTY::BINDING_ENERGY_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::ECCENTRICITY_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::ECCENTRICITY_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::SEMI_MAJOR_AXIS_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::SEMI_MAJOR_AXIS_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::ROCHE_LOBE_RADIUS_1_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::ROCHE_LOBE_RADIUS_1_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::ROCHE_LOBE_RADIUS_2_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::ROCHE_LOBE_RADIUS_2_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::MASS_1_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::MASS_ENV_1,
BINARY_PROPERTY::MASS_1_FINAL,
BINARY_PROPERTY::RADIUS_1_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::RADIUS_1_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::STELLAR_TYPE_1_PRE_COMMON_ENVELOPE,
STAR_1_PROPERTY::STELLAR_TYPE,
STAR_1_PROPERTY::LAMBDA_FIXED,
STAR_1_PROPERTY::LAMBDA_NANJING,
STAR_1_PROPERTY::LAMBDA_LOVERIDGE,
STAR_1_PROPERTY::LAMBDA_LOVERIDGE_WINDS,
STAR_1_PROPERTY::LAMBDA_KRUCKOW,
STAR_1_PROPERTY::BINDING_ENERGY_FIXED,
STAR_1_PROPERTY::BINDING_ENERGY_NANJING,
STAR_1_PROPERTY::BINDING_ENERGY_LOVERIDGE,
STAR_1_PROPERTY::BINDING_ENERGY_LOVERIDGE_WINDS,
STAR_1_PROPERTY::BINDING_ENERGY_KRUCKOW,
BINARY_PROPERTY::MASS_2_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::MASS_ENV_2,
BINARY_PROPERTY::MASS_2_FINAL,
BINARY_PROPERTY::RADIUS_2_PRE_COMMON_ENVELOPE,
BINARY_PROPERTY::RADIUS_2_POST_COMMON_ENVELOPE,
BINARY_PROPERTY::STELLAR_TYPE_2_PRE_COMMON_ENVELOPE,
STAR_2_PROPERTY::STELLAR_TYPE,
STAR_2_PROPERTY::LAMBDA_FIXED,
STAR_2_PROPERTY::LAMBDA_NANJING,
STAR_2_PROPERTY::LAMBDA_LOVERIDGE,
STAR_2_PROPERTY::LAMBDA_LOVERIDGE_WINDS,
STAR_2_PROPERTY::LAMBDA_KRUCKOW,

```
    STAR_2_PROPERTY::BINDING_ENERGY_FIXED,
    STAR_2_PROPERTY::BINDING_ENERGY_NANJING,
    STAR_2_PROPERTY::BINDING_ENERGY_LOVERIDGE,
    STAR_2_PROPERTY::BINDING_ENERGY_LOVERIDGE_WINDS,
    STAR_2_PROPERTY::BINDING_ENERGY_KRUCKOW,
    BINARY_PROPERTY::MASS_TRANSFER_TRACKER_HISTORY,
    BINARY_PROPERTY::STELLAR_MERGER,
    BINARY_PROPERTY::OPTIMISTIC_COMMON_ENVELOPE,
    BINARY_PROPERTY::COMMON_ENVELOPE_EVENT_COUNT,
    BINARY_PROPERTY::DOUBLE_CORE_COMMON_ENVELOPE,
    STAR_1_PROPERTY::IS_RLOF,
    STAR_1_PROPERTY::LUMINOSITY_PRE_COMMON_ENVELOPE,
    STAR_1_PROPERTY::TEMPERATURE_PRE_COMMON_ENVELOPE,
    STAR_1_PROPERTY::DYNAMICAL_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_1_PROPERTY::THERMAL_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_1_PROPERTY::NUCLEAR_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::IS_RLOF,
    STAR_2_PROPERTY::LUMINOSITY_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::TEMPERATURE_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::DYNAMICAL_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::THERMAL_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::NUCLEAR_TIMESCALE_PRE_COMMON_ENVELOPE,
    BINARY_PROPERTY::ZETA_STAR,
    BINARY_PROPERTY::ZETA_LOBE,
    BINARY_PROPERTY::SYNCHRONIZATION_TIMESCALE,
    BINARY_PROPERTY::CIRCULARIZATION_TIMESCALE,
    STAR_1_PROPERTY::RADIAL_EXPANSION_TIMESCALE_PRE_COMMON_ENVELOPE,
    STAR_2_PROPERTY::RADIAL_EXPANSION_TIMESCALE_PRE_COMMON_ENVELOPE,
    BINARY_PROPERTY::IMMEDIATE_RLOF_POST_COMMON_ENVELOPE,
    BINARY_PROPERTY::SIMULTANEOUS_RLOF

};
```

BSE Supernovae

```
const ANY_PROPERTY_VECTOR BSE_SUPERNOVAE_REC = {
    BINARY_PROPERTY::RANDOM_SEED,
    SUPERNOVA_PROPERTY::DRAWN_KICK_MAGNITUDE,
    SUPERNOVA_PROPERTY::KICK_MAGNITUDE,
    SUPERNOVA_PROPERTY::FALLBACK_FRACTION,
    BINARY_PROPERTY::ORBITAL_VELOCITY_PRE_SUPERNOVA,
    BINARY_PROPERTY::DIMENSIONLESS_KICK_MAGNITUDE,
    SUPERNOVA_PROPERTY::TRUE_ANOMALY,
    SUPERNOVA_PROPERTY::SUPERNOVA_THETA,
    SUPERNOVA_PROPERTY::SUPERNOVA_PHI,
    SUPERNOVA_PROPERTY::SN_TYPE,
    BINARY_PROPERTY::ECCENTRICITY_PRE_SUPERNOVA,
    BINARY_PROPERTY::ECCENTRICITY,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_PRE_SUPERNOVA_RSOL,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_RSOL,
    BINARY_PROPERTY::TIME,
    BINARY_PROPERTY::SUPERNOVA_STATE,
    BINARY_PROPERTY::UNBOUND,
    COMPANION_PROPERTY::RUNAWAY,
    COMPANION_PROPERTY::STELLAR_TYPE,
    SUPERNOVA_PROPERTY::STELLAR_TYPE,
    SUPERNOVA_PROPERTY::STELLAR_TYPE_PREV,
    COMPANION_PROPERTY::MASS,
    SUPERNOVA_PROPERTY::TOTAL_MASS_AT_COMPACT_OBJECT_FORMATION,
    SUPERNOVA_PROPERTY::CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    SUPERNOVA_PROPERTY::CO_CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    SUPERNOVA_PROPERTY::HE_CORE_MASS_AT_COMPACT_OBJECT_FORMATION,
    SUPERNOVA_PROPERTY::MASS,
    SUPERNOVA_PROPERTY::EXPERIENCED_RLOF,
    SUPERNOVA_PROPERTY::MASS_TRANSFER_DONOR_HISTORY,
    SUPERNOVA_PROPERTY::SPEED,
    COMPANION_PROPERTY::SPEED,
    BINARY_PROPERTY::SYSTEMIC_SPEED,
    SUPERNOVA_PROPERTY::IS_HYDROGEN_POOR,

};
```

BSE Pulsar Evolution

```
const ANY_PROPERTY_VECTOR BSE_PULSAR_EVOLUTION_REC = {
    BINARY_PROPERTY::RANDOM_SEED,
    STAR_1_PROPERTY::MASS,
    STAR_2_PROPERTY::MASS,
    STAR_1_PROPERTY::STELLAR_TYPE,
    STAR_2_PROPERTY::STELLAR_TYPE,
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_RSOL,
    BINARY_PROPERTY::MASS_TRANSFER_TRACKER_HISTORY,
    STAR_1_PROPERTY::PULSAR_MAGNETIC_FIELD,
    STAR_2_PROPERTY::PULSAR_MAGNETIC_FIELD,
    STAR_1_PROPERTY::PULSAR_SPIN_FREQUENCY,
    STAR_2_PROPERTY::PULSAR_SPIN_FREQUENCY,
    STAR_1_PROPERTY::PULSAR_SPIN_DOWN_RATE,
    STAR_2_PROPERTY::PULSAR_SPIN_DOWN_RATE,
    BINARY_PROPERTY::TIME,
    BINARY_PROPERTY::DT
};
```

BSE RLOF Parameters

```
const ANY_PROPERTY_VECTOR BSE_RLOF_PARAMETERS_REC = {
    BINARY_PROPERTY::RLOF_CURRENT_RANDOM_SEED,
    BINARY_PROPERTY::RLOF_CURRENT_STAR1_MASS,
    BINARY_PROPERTY::RLOF_CURRENT_STAR2_MASS,
    BINARY_PROPERTY::RLOF_CURRENT_STAR1_RADIUS,
    BINARY_PROPERTY::RLOF_CURRENT_STAR2_RADIUS,
    BINARY_PROPERTY::RLOF_CURRENT_STAR1_STELLAR_TYPE,
    BINARY_PROPERTY::RLOF_CURRENT_STAR2_STELLAR_TYPE,
    BINARY_PROPERTY::RLOF_CURRENT_SEMI_MAJOR_AXIS,
    BINARY_PROPERTY::RLOF_CURRENT_EVENT_COUNTER,
    BINARY_PROPERTY::RLOF_CURRENT_TIME,
    BINARY_PROPERTY::RLOF_CURRENT_STAR1_RLOF,
    BINARY_PROPERTY::RLOF_CURRENT_STAR2_RLOF,
    BINARY_PROPERTY::RLOF_CURRENT_COMMON_ENVELOPE,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR1_MASS,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR2_MASS,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR1_RADIUS,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR2_RADIUS,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR1_STELLAR_TYPE,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR2_STELLAR_TYPE,
    BINARY_PROPERTY::RLOF_PREVIOUS_SEMI_MAJOR_AXIS,
    BINARY_PROPERTY::RLOF_PREVIOUS_EVENT_COUNTER,
    BINARY_PROPERTY::RLOF_PREVIOUS_TIME,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR1_RLOF,
    BINARY_PROPERTY::RLOF_PREVIOUS_STAR2_RLOF,
    STAR_1_PROPERTY::ZETA_SOBERMAN,
    STAR_1_PROPERTY::ZETA_SOBERMAN_HE,
    STAR_1_PROPERTY::ZETA_HURLEY,
    STAR_1_PROPERTY::ZETA_HURLEY_HE,
    STAR_2_PROPERTY::ZETA_SOBERMAN,
    STAR_2_PROPERTY::ZETA_SOBERMAN_HE,
    STAR_2_PROPERTY::ZETA_HURLEY,
    STAR_2_PROPERTY::ZETA_HURLEY_HE
};
```

BSE Switch Log

const ANY_PROPERTY_VECTOR BSE_SWITCH_LOG_REC = {

BINARY_PROPERTY::RANDOM_SEED,
BINARY_PROPERTY::TIME

};

The default record specification can be modified at runtime via a logfile record spcifications file (program option *--logfile-definitions*).

Note that the BSE Switch Log file has the following columns automatically appended to each record:

- The constituent star switching stellar type: 1 = Primary, 2 = Secondary.
- The stellar type from which the star is switching.
- The stellar type from which the star is switching.

STAR_SWITCHING

Data type:	INT
COMPAS variable:	*derived from* BaseBinaryStar::m_Star1/m_Star2
Description:	The constituent star switching stellar type. 1 = Primary, 2 = Secondary.
Header String:	STAR_SWITCHING

SWITCHING_FROM

Data type:	INT
COMPAS variable:	*derived from* BaseStar::m_StellarType
Description:	The stellar type of the constituent star immediately prior to the switch.
Header String:	SWITCHING_FROM

SWITCHING_TO

Data type:	INT
COMPAS variable:	*Not applicable*
Description:	The stellar type to which the constituent star will switch (i.e. the stellar type immediately following the switch.
Header String:	SWITCHING_TO

These columns will always be automatically appended to each BSE Switch Log record: they cannot be removed via the logfile record specifications file.

Example Log File Record Specifications File

Following is an example log file record specifications file. COMPAS can be configured to use this file via the *logfile-definitions* program option.

This file (COMPAS_Output_Definitions.txt) is also delivered as part of the COMPAS github repository.

```
# sample standard log file specifications file

# the '#' character and anything following it on a single line is considered a comment
# (so, lines starting with '#' are comment lines)

# case is not significant
# specifications can span several lines
# specifications for the same log file are cumulative
# if a log file is not specified in this file, the default specification is used


# SSE Parameters

# start with the default SSE Parameters specification and add ENV_MASS

sse_sysparms_rec += { STAR_PROPERTY::ENV_MASS }

# take the updated SSE Parameters specification and add ANGULAR_MOMENTUM

sse_sysparms_rec += { STAR_PROPERTY::ANGULAR_MOMENTUM }

# take the updated SSE Parameters specification and subtract MASS_0 and MDOT

sse_sysparms_rec -= { STAR_PROPERTY::MASS_0, STAR_PROPERTY::MDOT }


# BSE System Parameters

bse_sysparms_rec = {                      # set the BSE System Parameters specification to:
    BINARY_PROPERTY::ID,                  # ID of the binary
    BINARY_PROPERTY::RANDOM_SEED,         # RANDOM_SEED for the binary
    STAR_1_PROPERTY::MZAMS,               # MZAMS for Star1
    STAR_2_PROPERTY::MZAMS                # MZAMS for Star2
}

# ADD to the BSE System Parameters specification:
# SEMI_MAJOR_AXIS_INITIAL for the binary
# ECCENTRICITY_INITIAL for the binary
# SUPERNOVA_THETA for Star1 and SUPERNOVA_PHI for Star1
```

```
bse_sysparms_rec += {
    BINARY_PROPERTY::SEMI_MAJOR_AXIS_INITIAL,
    BINARY_PROPERTY::ECCENTRICITY_INITIAL,
    STAR_1_PROPERTY::SUPERNOVA_THETA, STAR_1_PROPERTY::SUPERNOVA_PHI
}

bse_sysparms_rec += {                               # ADD to the BSE System Parameters specification:
    SUPERNOVA_PROPERTY::IS_ECSN,                    # IS_ECSN for the supernova star
    SUPERNOVA_PROPERTY::IS_SN,                      # IS_SN for the supernova star
    SUPERNOVA_PROPERTY::IS_USSN,                    # IS_USSN for the supernova star
    SUPERNOVA_PROPERTY::EXPERIENCED_PISN,           # EXPERIENCED_PISN for the supernova star
    SUPERNOVA_PROPERTY::EXPERIENCED_PPISN,          # EXPERIENCED_PPISN for the supernova star
    BINARY_PROPERTY::UNBOUND,                       # UNBOUND for the binary
    SUPERNOVA_PROPERTY::MZAMS,                      # MZAMS for the supernova star
    COMPANION_PROPERTY::MZAMS                       # MZAMS for the companion star
}

# SUBTRACT from the BSE System Parameters specification:
# RANDOM_SEED for the binary
# ID for the binary

bse_sysparms_rec -= {                      # SUBTRACT from the BSE System Parameters specification:
    BINARY_PROPERTY::RANDOM_SEED,          # RANDOM_SEED for the binary
    BINARY_PROPERTY::ID                    # ID for the binary
}

# BSE Double Compas Objects

# set the BSE Double Compact Objects specification to MZAMS for Star1, and MZAMS for Star2

BSE_DCO_Rec = { STAR_1_PROPERTY::MZAMS, STAR_2_PROPERTY::MZAMS }

# set the BSE Double Compact Objects specification to empty - nothing will be printed
# (file will not be created)

BSE_DCO_Rec = {}


# BSE Supernovae

BSE_SNE_Rec = {}            # set spec empty - nothing will be printed (file will not be created)


# BSE Common Envelopes

BSE_CEE_Rec = {}            # set spec empty - nothing will be printed (file will not be created)


# BSE Pulsars

# line ignored (comment). BSE Pulsars specification will be default

# BSE_Pulsars_Rec = { STAR_1_PROPERTY::MASS, STAR_2_PROPERTY::MASS }
```

```
# BSE Detailed Output

BSE_Detailed_Rec = {}          # set spec empty - nothing will be printed (file will not be created)
```