LISA BPS Common Output Format

Draft Proposal v. 0.2[1].

Luca Graziani[2] & Ruggero Valli[3],
and the LISA Synthetic UCB Catalogue Group
June 23, 2023.

1 Introduction

This document describes a common format for the output of binary population synthesis (BPS) codes agreed upon by the members of the LISA Synthetic UCB Catalogue Group. The goal of the format is to provide a common reference framework to describe the evolution of a single, isolated binary system or a population of isolated binaries.

1.1 Motivation

To provide a common output format across BPS codes participating in the LISA collaboration in order to:

- Simplify data processing, especially for large datasets.
- Easily and safely compare different BPS codes predictions.
- Minimize errors due to unit conversion, and avoid misinterpretation of similar physical quantities, by adopting a common definition.
- Provide a standardized framework of physical quantities facilitating the interpretation of BPS codes scientific outputs.

[1] Once approved by the collaboration the document will be submitted as White Paper on the ArXiV to receive a unique DOI. The collaboration will discuss the possibility to submit this work to a journal on Computational Astrophysics, in a different form. We propose to open a GitHub r CVS site to store software tools or code extensions as a free software accessible to everybody.

[2] `www.phys.uniroma1.it`, `luca.graziani@uniroma1.it`

[3] `www.mpa-garching.mpg.de`, `ruvalli@mpa-garching.mpg.de`

1.2 Leading Principles

The proposed common format is designed to adhere to the following general principles:

Progressive: the common format is designed to be forward/backward-compatible with its extensions and additions. This means that the common format is aimed at being capable to change in time in order to comply with the evolution of the BPS codes currently in the collaboration (and their future modifications), as well as to include new BPS codes.

Flexible: the common format can accommodate both specific and generic information, as well as missing data.

Unambiguous: the common format defines value, organization and labelling of all required data and its compliance as precisely as possible in order to minimize differences in its practical implementation.

Easy to implement: the proposed format, in its initial version, aims at being close to the data formats currently adopted by existing BPS codes.

To satisfy the previous requirements the format adopts a hierarchical organization of its values in stratified layers dubbed as L0 (common core variables), L1 and L2. Their definition is provided below and could change and vary across progressive versions of common format. As a rule, a file containing data defined as compliant with the common format should be provided by specifying the reference format version and the implemented level in the header.

2 Description of the format

The common format can be practically implemented in both hdf5 and ASCII csv file formats. In this version we describe its csv implementation. A ASCII csv file conforming to the present format is required to respect the following rules :

- Lines starting with the character `'#'` are called *free-format* or comment lines. Their number is unspecified and their content doesn't follow any specific format rule.
- Lines starting with a character different form `'#'` contain a sequence of fields separated by commas `','`.
- Quantities in double precision (i.e having a double value), are represented in exponential notation with a number of decimal digits ≥ 2

(e.g. 1.23e10 or 1.23456e10). The standard suggests a maximum number of digits < 5.

- Quantities of type string (not present in a comment line) can contain all printable ASCII characters, with the exception of `','` and `'#'`, which are defined as reserved characters.

A common format file is composed of three parts: a **the Free-Format section** (§2.1), the **the Header** (§2.2) and **the Table of Values** (§2.3).

2.1 The Free-Format section

The Free-Format section is intended to store simulation specific parameters as well as comments and information regarding the run generating the stored dataset. The section is composed by comment lines only, as defined above. It ends at the first line of the file that doesn't start with `'#'`. The user can freely choose both content and number of lines of a Free-Format section.

Examples

Some possible use-cases for the free format section are listed below.

- Each simulation run is generally identified by a timestamp, a specific version of the adopted BPS code binary or even a build-release of the code. A simulation unique ID or name is also often adopted to uniquely label different runs. All the above information is likely stored in the Free-Format section of the file if the fields defined in the Header Section are not sufficient.
- When performing several runs for a parameter exploration study, the Free-Format section can be used to store the adopted ranges of values.
- The exact value of physical constants used in the code can be stored in the Free-Format section to guarantee consistency with post-processing operations requiring unit conversions.

NOTE: It is not advisable to write in the Free-Format section the values of *all* the set-up parameters adopted in the run, because many of them are likely to be specific to a particular code and it would considerably clutter the header. I this sense the free-format section should not be considered as a substitute of proper run documentation or an associated log file. The best way to make simulation results reproducible is to provide your run-scripts, the version of a code binary as well as the code documentation.

2.2 The Header

The Header section is intended to provide a minimum set of parameters uniquely associating the stored data with the common format version as well as the generating BPS code binary. It begins at the first line that doesn't start with `'#'` and it is composed of 3 lines. The first 2 lines contain the same number of columns, each containing a field. The first line contains the unique labels of the fields, while the second line contains their specific values, appearing in the same order as the labels they correspond to. The required header fields are described in Table 1. The third line of the header contains the unique labels of the variables describing each binary system evolution according to the specific implemented level (either defined in L0, L1 or L2, see table 2). Their physical corresponding values are listed in

columns, in the same order of the Table of Values.

Examples

An example of a Header compliant with the standard is the following:

```
1 cofVer,cofLevel,cofExtension,bpsName,bpsVer,contact,NSYS,
    NLINES,Z
2 1.0,L0,,ComBinE,1.0.0,pippo@pluto.com,0.01,1000,12345
3 ID,UID,time,event,semiMajor,eccentricity,type1,mass1,radius1
    ,Teff1,massHeCore1,type2,mass2,radius2,Teff2,massHeCore2
```

Forward-compatibility: the Header

To insure forward-compatibility, the only requirement is that **all mandatory labels described in Table 1 are present in the Header**. The order in which they appear is not fixed. Moreover, additional fields that are not described in Table 1 are ignored. In this way, future extensions of the format that add new header fields will be compatible with parsers that are compliant with older versions.

Data integrity check

The two header fields `NSYS` and `NLINES` can be used as a simple check of the integrity of the file. Ensuring that the number of systems stored in the file and that the number of lines in the file correspond to the values reported in the header helps identifying problems during the copy or the manipulation of the file.

2.3 The Table of Values

The evolution of binary systems, as collection of their physical states, is contained in the Table of Values. After the third line of the header, the file contains a table of values corresponding, in the same order, to the columns listed at line 3 of the header. Each row describes the state of a binary system at a given point in time, as a collection of properties of the binary and of its components at fixed time. The states pertaining to the same systems must appear in contiguous rows and are likely ordered chronologically. If two consecutive lines pertain to the same system, the value of the `time` column should increase with the line number, unless the time between two states goes below the resolution of the code. To guarantee that consecutive states

Label	Description	Mandatory?	Type	Example
`cofVer`	Common Format Version	Y	string	1.0
`cofLevel`	Common Format Level (§2.3.1)	Y	string	L0
`cofExtension`	Name of the format extension (§4)	Y	string	myExtension
`bpsName`	BPS code used to generate the data	Y	string	SEVN
`bpsVer`	BPS code version	Y	string	1.0.0
`contact`	Contact person or email	Y	string	pippo@pluto.com
`NSYS`	Number of binary systems in the file	Y	long	1000000
`NLINES`	Total number of lines in the file	Y	long	12345678
`Z`	Initial absolute metallicity.	N	double	1.00e-2

Table 1: Fields of the Header section. Note that different `bpsName` labels should indicate different codes or different code branches developed independently, otherwise the label version should mark different code releases.

are uniquely identified, this standard assumes that the line printing order also defines their logical priority. Table 2 describes the columns required by the format and their organization in levels. The columns must be all present in the file once the implemented level is established, but some of the columns may be left empty if not provided by the code (see box *Undefined and missing values*). The format also allows comment lines between binary system lines, while comment lines between state lines are forbidden. Comment lines separating binaries could be used, for example, to specify a different value of the assumed binary metallicity. In this case the label `Z` is missing in the header file, while repeated in a comment line before a new system lineset. It should be noted though, that for large datasets a more scalable solution should be adopted; see Section 3 for more details.

2.3.1 Levels

The meaning of the levels is as follows:

- Level 0 (L0) is referred to as Common Core Level, indicating the minimum set of physical variables describing a binary system physical status at fixed time (i.e. a binary state). All the BPS codes participating the collaboration share values of L0 so that their physical outcome can be safely compared once provided in the common format.
- Level 1 (L1) contains all the variables of L0, plus a number of variables shared by the largest group of participating BPS codes. Not all codes

are expected to provide L1 values so the comparison across codes is limited with respect to L0. L1 also serves to stimulate the development of missing features in BPS codes missing these quantities.

- Level 2 (L2) groups all the quantities in L0 and L1, plus a number of physical values provided by some of the participating BPS codes without requirement of a common intersection. For specific applications or analysis they could be necessary.

Undefined and missing values

Unless differently specified, when the value of a field is not defined, it is assigned the value `NaN`. When the value of a field is missing/unknown/not reported, the filed is left empty (i.e. two contiguous comma separators).
For instance, when a binary system has been unbound, its orbital parameters (`eccentricity`, `semiMajor`, etc.) are not defined anymore, and those fields will be assigned the value `NaN`. On the other hand, if the BPS code does not provide the eccentricity in the output, the component may or may not have a defined eccentricity, but we don't know its value. The field will then be left empty.
Other examples of quantities that may be not defined are the properties (mass, radius...) of an object that doesn't exist anymore (type = -1).

ID and UID

There are two kinds of IDs. The *progressive system ID* (`ID`) is an integer number that is assigned progressively to the systems. The first system appearing in the file will have `ID` 1, the second will have `ID` 2 and so on until the last system with `ID` `NSYS`. The `ID` is an easy way to identify a system within the same file. For example, for a fast scanning of the systems, this value can be used to select ranges (e.g. 100-105) or to quickly find a specific system ID to debug.
The *unique system ID* (`UID`) can instead be any string and can be used to identify systems across different files. The only requirement for the `UID` is that two systems cannot have the same `UID` within the same file. It can be used, for example, to match the system described in the standard format with the analogous one in the original file of the specific BPS code output.

Label	Description	Unit	Variable Type
	L0		
ID	progressive system ID		long
UID	unique system ID		string
time	time	Myr	double
event	event type[a]		string
semiMajor	semi-major axis	$R_\odot$	double
eccentricity	orbital eccentricity		double
type1	type of object 1[b]		string
mass1	mass 1	$M_\odot$	double
radius1	radius 1	$R_\odot$	double
Teff1	effective temperature 1	K	double
massHecore1	He core mass 1	$M_\odot$	double
type2	type of object 2[b]		string
mass2	mass 2	$M_\odot$	double
radius2	radius 2	$R_\odot$	double
Teff2	effective temperature 2	K	double
massHeCore2	He core mass 2	$M_\odot$	double
	L1		
envBindEn	envelope binding energy	erg	double
massCOCore1	CO core mass 1	$M_\odot$	double
massCOCore2	CO core mass 2	$M_\odot$	double
radiusRL1	radius of the Roche Lobe 1	$R_\odot$	double
radiusRL2	radius of the Roche Lobe 2	$R_\odot$	double
period	orbital period	day	double
luminosity1	bolometric luminosity 1	$L_\odot$	double
luminosity2	bolometric luminosity 2	$L_\odot$	double
	L2		
dMdt1	mass gained/lost by 1	$M_\odot/yr$	double
dMdt2	mass gained/lost by 2	$M_\odot/yr$	double
jOrb	orbital angular momentum	$M_\odot\ {R_\odot}^2$/day	double
spin1	abs. value of spin 1	$M_\odot\ {R_\odot}^2$/day	double
spin2	abs. value of spin 2	$M_\odot\ {R_\odot}^2$/day	double
omega1	angular velocity 1	day^{-1}	double
omega2	angular velocity 2	day^{-1}	double
Hsup	H surface mass fraction		double
Hesup	He surface mass fraction		double
Csup	C surface mass fraction		double
Nsup	N surface mass fraction		double
Osup	O surface mass fraction		double
spectralType1	spectral type 1		string
spectralType2	spectral type 2		string

Table 2: The columns present in the table. [a]See Section 2.3.2. [b]See Section 2.3.3.

Forward-compatibility: the Table of Values

To insure forward-compatibility, the only requirement is that **all the column described in Table 2 are present in the Table of Values**. The labels are case sensitive and must be written exactly as in this document. The order in which they appear is not fixed. Moreover, additional columns that are not described in Table 2 and not codified in an extension are ignored. Additional labels are thus not forbidden, while not accounted for by the standard. In this way, future versions of the format that add new columns will be compatible with parsers that are compliant with older versions.

2.3.2 Component types

A component type is a numerical label that describes the evolutionary state of one of the two components of the binary system. The description can be generic (e.g. star) or detailed (e.g. asymptotic giant branch star), and this is achieved by placing the labels in a tree-like structure, where parents are more generic than children and leafs are the most specific labels. The numerical value of the label mimics the structure of the tree. The leftmost digit indicates the most generic branch, up to the rightmost digits that indicates the most specific leaf. For example, the numerical label for a first giant branch star is 123, where 1 in the first digits indicates that it is a star, 2 in the second digit means that it has a hydrogen envelope and 3 that it is currently burning hydrogen in a shell.

At each level on the tree, the number 9 is reserved for 'other', and can be used in case more than 8 elements are needed at a particular level. The additional ones can be labelled 91, 92.. etc. Future extensions to the format will preferably follow the same labelling pattern.

The following numerical labels are supported by this version of the common format. Any label that is not in the following list shall be considered equivalent to -2 (unknown/missing).

- 1 - Star: this category contains all objects that are undergoing some form of nuclear fusion in their interior and can be classified as stars
  - 11 - Protostar: an object of stellar mass that has not reached the zero age main sequence.
  - 12 - Star with a hydrogen envelope
    - 121 - Main sequence (or core hydrogen burning)

        - 122 - Hertzsprung gap
        - 123 - First giant branch (or shell hydrogen burning)
        - 124 - Core helium burning
        - 125 - Asymptotic giant branch (or shell helium burning)
            - 1251 - Early asymptotic giant branch
            - 1252 - Thermally pulsing asymptotic giant branch
    - 13 - Helium star: star that is completely depleted of a hydrogen envelope
        - 131 - Helium main sequence
        - 132 - Helium Hertzsprung gap
        - 133 - Helium first giant branch
    - 14 - Carbon star: star that is completely depleted of both hydrogen and helium envelope
    - 15 - Chemically homogeneous star: star that maintains a negligible internal chemical gradient due to efficient chemical mixing processes.
- 2 - White dwarf
    - 21 - Helium white dwarf
    - 22 - Carbon-Oxygen white dwarf
    - 23 - Oxygen-Neon white dwarf
- 3 - Neutron star
- 4 - Black hole
- 5 - Planet
- 6 - Brown dwarf
- 7 - Thorne-Żytkow Object: is a possible result of the merger of a star and a neutron star. A star with an accreting neutron star in the core.
- 9 - Other: anything that does not fit in any previous category.
- -1 - Massless remnant: this object doesn't exist anymore. e.g. it has exploded or has been disrupted or has merged with the companion. This label serves as placeholder.
- -2 - Unknown: the component type of the object is not specified. This information is missing.

2.3.3 Events

A state event is a numerical label that describes what happened in the system that triggered the output of the current line (or equivalently the output of an interesting state). The numbering system is similar to the one used for the component type.

Hereafter, the character $*$ stands for 0, 1, 2 or 3 with the meaning

- 0 - not specified
- 1 - component 1
- 2 - component 2
- 3 - both components

The following numerical labels for events are supported by this version of the common format. Any label that is not in the following list shall be considered equivalent to -2 (unknown/missing).

- 1$*$ - Component $*$ changes type
- 2$*$ - Component $*$ goes supernova
  - 2$*$1 - runaway thermonuclear explosion in degenerate matter (type Ia supernova). 231 for a double degenerate scenario, while in a single degenerate scenario $*$ is the index of the accretor.
  - 2$*$2 - Core collapse supernova
  - 2$*$3 - Electron capture supernova
  - 2$*$4 - Pair-instability supernova
  - 2$*$5 - Pulsational pair-instability supernova
  - 2$*$6 - Failed supernova (direct collapse into a black hole)
- 3$*$ - Component $*$ overflows its Roche lobe
- 4$*$ - Component $*$ goes back into its Roche lobe (end of stable mass transfer or end of a contact phase)
- 5 - the surface of the two components touch.
  - 51$*$ - Component $*$ engulfs the companion, triggering a common envelope (513 is the double common envelope).
  - 52 - the two components merge.

    - 53 - the system initiates a contact phase.
    - 54 - the two components collide at periastron.
- 8 - terminating condition reached, evolution is stopped. Only the last line of the evolution of a system can have an event with 8 as first digit. If more than one condition applies at the final step, any of them can be provided.
    - 81 - max time reached
    - 82 - both components are compact remnants
    - 83 - the binary system is dissociated
    - 84 - only one object is left (e.g. due to a merger or because the companion has been disrupted)
    - 85 - nothing left (both components are massless remnants)
    - 89 - other: a terminating condition different from any previous one
- 9 - other: any event that does not fit in any previous category.
- -1 - no notable events happened in this time step. This can be used when the output of the code is not based on events, but is given at fixed time intervals.
- -2 - unknown: the event is not specified. This information is missing.

3 Log file for system-specific parameters

Specific applications focused on parameter exploration could produce output files mixing binary systems having a different set of physical parameters or exploring combinations of physical assumptions. In this scenario, every system in a given file is in principle produced by a set of different parameters. For example, a realistic sample of Galactic binaries, will have a large number of systems at different initial metallicity. The conditions of each binary should be careful traced for their comparison and physical interpretation, while the size of the dataset certainly impedes their description in comment lines, although allowed by the standard.

As scalable workaround, the present version of the standard format allows the existence of a log file, having the same name of the data file and ".log" extension, in which the ID and UID are associated with all the required parameters.

Label	Description	Unit	Variable Type
`ID`	progressive system ID		long
`UID`	unique system ID		string
`semiMajor`	semi-major axis	$R_\odot$	double
`eccentricity`	orbital eccentricity		double
`type1`	type of object 1		string
`mass1`	mass 1	$M_\odot$	double
`Z1`	absolute metallicity 1		double
`type2`	type of object 2		string
`mass2`	mass 2	$M_\odot$	double
`Z2`	absolute metallicity 2		double

Table 3: An example of columns for the initial condition file

The present document does not standardize the content of the log file, which is left free to specific needs of BPS code implementors.

As a usage example, a log file could be adopted to store the initial conditions generating a binary system set. Table 3 provides an example of initial conditions storable in a .log file.

4 User or BPS-specific Extensions

The present version of the common format is designed to be compatible with user-specific or BPS-specific extensions. An extension could be required, for example, when a combination of values does not conform to either L1 or L2 or when it is convenient to mix a certain common format level (e.g. L0) with BPS-specific outputs. In this case the data file will be declared as conform to L0, extended with a documented set of parameters. When a code extension becomes adopted by all BPS codes, the extension implementors could consider asking its inclusion at L0 layer of the common format and the extension becomes part of a new format release. A widely used extension could also be included in either L1 or L2 depending on its popularity across BPS codes.

The following classes of extensions are allowed in the present version:

- addition of fields in the Header,
- addition of new columns in the Table of Values,
- definition of new component types and event types

4.1 Defining an Extension

Users or BPS code implementors can define their own extensions of the present format by writing a document containing

1. the name of the extension,
2. the unique labels of the new columns/header fields, their variable type and a short description of their meaning.
3. the unique numerical labels of the new component types/event types and a short description of their meaning.

This document will be shared as additional file and to anyone that need to use the dataset implementing the described extension. The common format does not specify the extension types, values or events but requires them to be written as columns/values successive to the last value of the adopted layer. File parsers will safely work with columns codified by the layer format (e.g. L1) and should be modified to understand extension columns or labels.

4.2 Using an Extension

To use an extension

1. write the name of the extension in the field `cofExtension` in the Header. When no extension is used the field is left empty.
2. add the labels of the new columns/header fields to the Header.
3. fill those columns/header fields with the new defined quantities.
4. use the new component types/event types labels in the appropriate column, when necessary.