BOLTS Specification Version 0.2

The BOLTS library consists of several different components, processes and conventions and a number of precisely defined concepts. The purpose of this document is to specify them.

Parts, Classes, Standards and Collections

In this section some concepts are defined, that will be used later in the specification of blt-files.

Parts

A BOLTS repository contains a data about objects, that are in some way useful in CAD. A part is a object that can be described by a set of dimensions. E.g. a piece of paper is described by its width of 210mm and its height of 297mm.

Classes

However, one often encounters parts that are very similar to each other, for example just differing in their dimensions. These parts can then be more efficiently described as a class of parts. To continue with the paper example, a class describing pieces class could also contain a part with width 100mm and height 100mm.

In blt-files classes of parts are described, because this case is so frequent. Because some parts are one of a kind, it is not uncommon to have classes that contain just a single part.

The classes of technical parts that BOLTS deals with are often specified by standards issued by standardization bodies. blt-files allow to specify this for a class.

Collections

It is convenient to organize the classes that we have in a bolts repository in collections. A collection usually has one or few authors, the parts contained are in some sense related to each other and all the data of a collection is under the same licence.

Each collection has a one word identifier, the collection id. The collection id is the filename (without extension) of the blt-files with the information about the collection. For more details information see blt-files.

The repository

A BOLTS repository is directory structure with a certain layout. It contains all the data and metadata, as well as backend specific extradata. The root directory contains at least the "data", "drawings" and "output" directories, plus optionally a number of backend directories. See list-of-backends for details.

The data directory

The data directory contains a number of blt-files, one for each collection. These files contain most of the backend independent information about the parts in the repository.

The drawings directory

The drawings directory contains a number of subdirectories, usually one for each collection. In each of these directories, drawings of the parts are stored, that illustrate the geometries of the parts and the meaning of the parameters.

The format of these drawings is not specified, but svg or png files are recommended.

The output directory

The distributions generated by the backends end up in subdirectories of the output directory. Its contents should not be modified, as they get overwritten, when the distribution is regenerated.

The backend directories

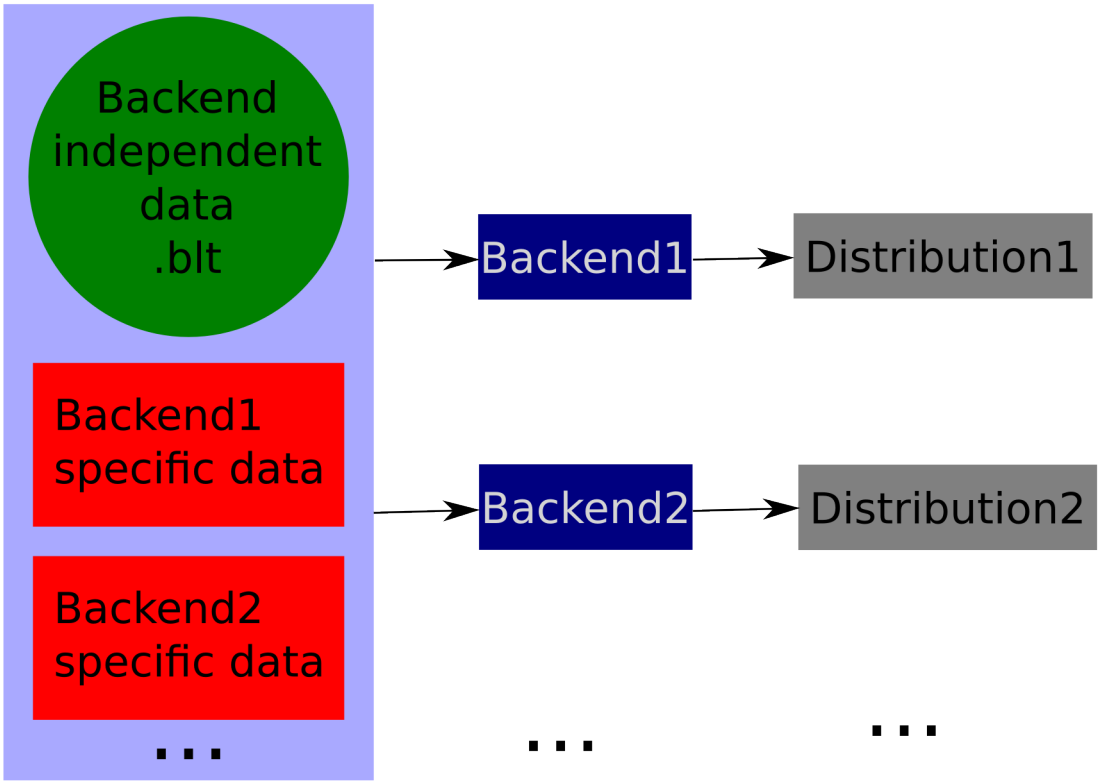
Each backend directory can contain files and directories with additional informations, which are required by the backend. Details are specified by the backend.

Backends, Processing and Distributions

BOLTS tries to separate the backend independent data and metadata from the backend specific data. The former is held in blt-files, the latter in base-files.

Backend

A backend is a process that uses backend independent data and backend specific data as input and has one or several files as output. The output is called the distribution.



A example for a backend would be a process that uses the backend indepent data about parts, their geometries and dimensions together with a number of templates and stylesheets and produces a set of HTML pages with a nicely rendered, browsable description of the parts. Other backends could produce data that is suitable for use in specific CAD applications.

Often backends are relatively independent and do use only their own backend specific data. However, this is not necessarily so, a backend may also use backend specific data of other backends. For example the HTML backend described above might add informations about how to use a part in a specific CAD application, and this might require to access the backend specific data of the backend for this CAD application. However, such cross backend dependencies should be made optional, to keep every backend functional independently of other backends.

There is a list-of-backends, where more informations about the available backends can be found and their base-files are specified.

Backend independent data - the blt file

The backend independent data is stored in yaml files with the extension .blt with exactly one YAML document and containing an associative array with the following keys:

- collection: mandatory. The collection-header
- classes: mandatory. An list of class-element as values.

The filename without the .blt extension is called the collection id. Collection ids are one word identifiers, which must be unique within the repository. This class id is used as a way to refer to the class, when the standard field is not set. They should contain only letters, numbers and underscores, and should be descriptive, as they may be visible to the user. Some names can not be collection ids: common, gui

Collection header

The collection header is an associative array that holds general informations regarding the collection. It contains the following keys:

- name: optional, string. A name for the collection.
- description: optional, string. A description of the contents of this collection.
- author: mandatory, string or list of strings. The author of this collection with e-mail in <>. If multiple authors contributed significantly to this collection a list of authors may be given.
- license: mandatory, string. The name of the license for this collection and a URL pointing to the full text of the license enclosed in <>.
- blt-version: mandatory, number. The version of the blt format this collection follows.

Class element

A class element is an associative array that holds information about a class. It has the following keys:

- id: mandatory, string. The id of the class. Class ids are one word identifiers, which must be unique within the repository. This class id is used as a way to refer to the class, when the standard field is not set. They should contain only letters, numbers and underscores, and should be descriptive, as they may be visible to the user.
- naming: mandatory, naming-element. A naming convention for the parts of this class.
- drawing: optional, string. Filename of a drawing for this class.
- description: optional, string. A short description of the class.
- standard: optional, string or list of strings. The name of the standard, if class is standardized. In the case of several identical standards, a list of names can be given.
- status: optional, string. This can be used to indicate the status of the standard. Possible values are "active" and "withdrawn", if absent, "active" is assumed.
- replaces: optional, string or list of strings. Standards that are superseded by this standard.
- parameters: optional, parameter-element: Parameters for this class.
- url: optional, string or list of strings. A url with relevant information regarding the parts of this class. For example a link to a vendor, or to the specifying standard. In the case of several identical standards, a list of urls has to be given.
- notes: optional, string. Notes for this class. Can be used to formulate questions or additional information.

- source: mandatory, string. A short description where the informations for this class originate. Should contain a URL if possible.

Parameter element

A parameter element is an associative array that holds information about the parameters of a part. This information is used when doing parameter-collection. Parameters are usually dimensions, but are not restricted to be. The following keys are contained in a parameter element.

- literal: optional, associative array. This array has as its key the parameter names of the literal parameters, as values the corresponding values. Literal parameters are rarely used.
- free: optional, list. This list contains the names of the parameters that are not assigned a value. Usually the user will provide this value.
- tables: optional, table-element or list of table-elements. This array contains tabular data. Often the table index will be a free variable, for details see table-element and parameter-collection.
- types: optional, associative array. Contains as keys parameter names, as values their respective types. Possible types are: "Length (mm)", Length (in)", "Number", "Bool", "Table Index", "String". If no entry is present for a parameter, "Length (mm)" is assumed.
- defaults: optional, associative array. This array contains a default value for every free parameter. If absent, the default value defaults to the type specific values given in the table below.

Type	Default Value
Length(mm)	10
Length(in)	1
Number	1
Bool	False
Table Index	""
String	""

Some parameter names are forbidden: standard.

Table element

Tables of data are very common in standards and very useful for specifying a class of parts. A table element describes a table of values, where the row is specified by the value an index parameter, and each column contains the value for a parameter. A table element is an associative array that has the following keys:

- index: mandatory, string: name of the index parameter. Has to specified to be of type "Table Index" in the parameter-element.
- columns: mandatory, list: list of parameter names corresponding to the columns of the table.
- data: mandatory, associative array: The keys are possible values of the index parameter, the values list of values compatible with the types of the parameters specified in columns.

Naming element

The name of a part should be precise enough to completely describe it, and therefore depends on the values of (some of) the parameters. A naming element is an associative array that holds information about the name of the parts of a class. It has the keys:

- template: mandatory, string. A name template, can contain placeholders for strings "%s" and numbers "s".

- substitute: optional, list. List of parameter names and that should be filled in for the placeholders in the template. If missing defaults to empty list. Besides the parameter names from the parameter-element, also the special name "standard" can be used.

Parameter Collection

Parameter Collection is the process of assigning a value to each parameter. The set of all parameters is found by collecting parameter names from the fields of a parameter-element:

- The keys of the literal field.
- The items of the free field.
- The index field of the table-element s in the tables field.
- The columns field of the table-element s in the tables field.

It is an error condition if there is a parameter name present as a key in the types field, that is not in the set of all parameters.

Then a value is assigned to each parameter. This can happen by:

- A literal value given in the literal field
- User or external input for parameters listed in the free field
- Table lookup for parameters listed in the columns field of a table-element

It is an error condition if a parameter is not assigned a value or if there are more than one way to assign a value.

For example are the parameter values collected in this way used (among other properties) to populate the template given in the naming-element.

Base Files

Base files are yaml files, that store backend specific data about additional files for a collection. They consist of a list of base-file-element.

Base file element

A base file element is an associative array containing at least.

- filename: The filename of the
- type: A string describing the type of file, this is specific to the backend.

List of Backends

OpenSCAD

The files containing informations required by the OpenSCAD backend as well as the generated files (the distribution) reside in the backend directory with the name openscad.

This backend directory contains a folder for each collection that contains files related to this collection, and the folder is named like the collection-id.

The OpenSCAD backend generates a OpenSCAD library with modules for all parts from all collections for which the base modules are specified. Base modules are openscad modules that take as parameters a subset of all the parameters of the part (see parameter-collection), and construct the part according to these dimensions.

These modules can be placed in one or several files residing in the respective collection directory within the openscad backend directory. Each files is specified with a base-file-element containing the following keys and values:

- type: module
- modules: a list of base-module-element

Common files that contain code that is used in several collections or base files can be placed in a folder called common in the openscad backend directory. They will be automatically included into the main file and copied to the distribution.

Base module element

A base module element is a associative array describing a openscad base module with the following keys:

- name: name of the base module
- arguments: a list with the arguments that need to be supplied to the module. Can be a subset of the parameters of the class, see parameter-collection.
- ids: a list of class ids, for which code should be generated using this base module.