# **Brightway Enhancement Proposal (BEP)**

|  |  |
| --- | --- |
| Number | 1 |
| Title | Storing & using LCIA method (family) @ MultiLCA |
| Status | Draft |
| Type | Proposal |
| Proposed By | Marc Van der Meide, Jonathan Kidner, Victor Tulus, Cecilia Salah, Margarita Charalambous,Sidi Peng, Guillaume Batôt, Sander Van Nielen |
| Editor | All |
| Created | 2022-09-30 |
| Last updated | 2022-09-30 |
| Version | 1 |

## 

## 

## **Abstract**

LCA practitioners usually want to do LCA calculation with multiple impact categories (for example the 18 impact categories under ReCiPe). At present the function multi\_lca in brightway currently performs calculations with multiple methods after the user enters a tuple of complete method names. In addition, the data of LCIA methods are complicated to access, which hinders the users ability to understand, modify or add characterization factors. Therefore, we proposed a new interface design for multi-LCA, which would support the user entering only a method name (like ReCiPe), in addition to full specifications. Also, we compare different alternatives for CF data storage. The user preference being for centrally stored data with safe, reliable, and efficient data management.

## **Motivation**

Currently, the data of LCIA methods are very complicated to access, since it is not centrally stored. The metadata is stored in a json file and the LCIA methods are stored in Brightway (1) in different folders, processed version and intermediate version, (2) as multiple .npy files, one per method. This current structure has clear elements that make it more complicated for users to interrogate the data by hand and can also lead to errors through a more complicated linkage structure between the metadata file and low-level data files. Additionally, this structure does not lend itself naturally to extend the usage of the LCIA methods - incorporating multiple different levels, or multiple elements within the LCIA calculations.

## **Proposal**

**Interface design for multiple LCIA**

The MultiLCA class should support the user entering only the method name. The latter should be a string, that auto-completes once the user starts writing the name of the method of interest (e.g., the user can write “ReCiPe 2016” and the interface will propose a list of all unique entries that start by “ReCiPe 2016”). The code in the background should then be able to detect if the method entered as an argument for the MultiLCA class is a string or a list. If it is a string, it will call a function on the backend that lists all the tuples associated with the method in question. If it is a list, it will operate as it does now. This would avoid breaking the code of users of previous versions of Brightway.

**Data storage**

We propose to change the data storage format in Brightway. We believe that we should have a systematic way of storing data in secure and reliant formats. We see the benefits of employing SQL databases, or employing the ILCD format. This would enable for a more structured way of storing methods data, that would be compatible with different softwares, enabling consistency and reliability when employing specific methods in an LCA study.

Moreover, independently from the above point, we suggest changing the current format, to centralize the data for a single method i.e., one method that contains X impact categories should be stored as a single file, instead of X different files. In this way, when a method is updated, only one file is updated instead of X files.

### **Rationale**

**Interface design for multiple LCIA**

We believe that this class should be able to calculate all impact categories of a method, by only reading the method’s name. We would, therefore, support the development of a function that could be implemented from the MultiLCA class. This function should take a list of tuples associated with a given method and/or selected impact categories within a given method. A further point regarding the MultiLCA class interface is for backwards compatibility. To maintain this an additional check step would be required, enabling the user to provide a string (as an alternative to a list of tuples). If the method is provided with a string the method should proceed as currently specified, otherwise this new function to get the list of tuples.

**Data storage**

We have extensively discussed the different ways of storing data, especially thinking of a way to centralize the data in a robust way.

We see the benefit of utilizing SQL databases to store data, in a centralized way. The new entries can be added to the database, without a risk of overriding the previously existing data. This enables the storage of different data types, which can be beneficial for this type of application. Information about the different methods, such as characterization factors, units, method description, biosphere flows used, and author data are important for users, and should be secure with limited opportunity to be overwritten.

It makes sense to have a synthetic identifier to filter per method and version (the impact categories should not be considered as individual methods, but the flexibility to filter by impact category is nice to compare the results from the different methods to calculate the same indicator e.g., climate change).

## **Discussion**

**Data storage**

The points that came up and were considered important when comparing different data storage formats were the following:

* Data transparency: the chosen data format should enable easy accessibility by users
* Data reliance: the data storage should allow for analyses and considerations to be comparable among studies: enabling easy data manipulation is in principle viewed as a con; it should be possible to add new methods to the table, but not edit previously existing methods (changing characterization factors data of method X defeats the purpose of using this method; the better and more reliant approach would be to create a new method Y with the new characterization factors, defined by the user)
* Data portability: the data should allow to be easily transported and compatible with different softwares
* Data security: this is a complicated issue, users may occasionally require the ability to change values, however this is not assumed to be a common use case

### **Pros and Cons**

**Interface design for multiple LCIA**

Here we present pros and cons of utilizing the new functionality of the MultiLCA class.

| Pros | Cons |
| --- | --- |
| The user does not have to exhaustively look for all tuples of all impact categories of the method that they want to use  The user will have an option to check the list of all impact categories being calculated, displayed in a dictionary | The user has to provide the exact name of the method (case sensitive), which still requires a pre-understanding of the methods list |

**Data storage**

In the pros and cons table, we present an overview of the different alternatives for data storage that we have discussed.

|  | Pros | Cons |
| --- | --- | --- |
| SQL | Strict structure  More secure (in terms of data manipulation)/rigidity  Graphical tools exist to query and display SQL data in a very nice way  Powerful filtering function: The user would be able to enter the name partially, and complete if needed (as it is done in Activity Browser to filter activities and methods) <- can be implemented in any package?  Flexibility to store data types  Centralization of data  All the activities are already stored in a SQL database (Ecoinvent) | Less intuitive to export to other softwares  Less user-friendly (if user wants to open the file and check the data) - .csv is better for new users  Added layer of complexity  Not a portable infrastructure; to exchange data, you also need to exchange the code to access the databases |
| Data packages | More user flexibility/free access (we can store any type of data file)  Easy to access the data in those packages (good for transparency reasons) | Data is not stored in the same format (not accessible in the same way by code; user has to have a clear understanding of all file types and how to access them in the code)  Core data can be manipulated |
| Excel/csv files | Easily human readable  Easily modifiable data | Inefficient in memory for storing data  Less secure (easily manipulated, more open to human manipulation/mistakes)  Less reliant: changing previously existing data |
| Json files | Very flexible data storage/types  Human readable/transparent format |  |
| ILCD type | LCIA methods dedicated file, with all the metadata and the characterization factors data, coupled to the method id, in one single place (xml structure)  1 object that the user can import into the LCA software and access from there  Single artifact that is portable across different systems and very easily accessible by users (each impact categories )  Option to manipulate the characterization factors of the method and change them in the source code (pro: if you spot errors or inconsistencies you can fix them) | Every category is defined as a different method  Option to manipulate the characterization factors of the method and change them in the source code (con: because then the studies are not standardized and not comparable anymore) |

### **Alternatives**

**Data storage**

In consideration of the alternative data formats some of the key factors that we have identified to discriminate between the solutions is data transparency. An additional argument involved is with the distribution of data and the preferred solution of using few separate files for storage. No clear solution for this is ready and decisions on this matter need to be taken in subsequent steps.

### **Test plan and results**

**Interface design for multiple LCIA**

We developed a function that lists all tuples associated with a defined method. It can be found in this Github repository: <https://github.com/ceciliasalah/brightcon2022-hackathon/blob/main/adapt4multiLCA.py>

## 

## **BEP metadata**

**Databases for data storage**

SQLite proposal:

We propose implementing 3 tables that are connected by the ID\_meta of the method in question. They would store data in the following way:

1. Flows table:
   1. Flow ID
   2. Biosphere matrix entries
   3. Dependencies
   4. ID\_meta
2. Meta table
   1. ID\_meta
   2. Name
   3. Description
   4. Metadata fields (blob? Some other musts?)
3. Hierarchy
   1. ID\_hrc
   2. ID\_meta
   3. Parent
   4. depth

Flows table

| Flow ID | Biosphere matrix entries | Dependencies | ID\_meta |
| --- | --- | --- | --- |
|  |  |  |  |

| ID\_meta | Name | Description | Metadata fields |
| --- | --- | --- | --- |
|  |  |  |  |

| ID\_hrc | ID\_meta | Parent | Depth |
| --- | --- | --- | --- |
|  |  |  |  |

**Alternative proposals:**

General REQUIRED features:

* Simpler data storage using single .npy/.csv/.xls/.ilcd/.json files containing the characterization factors for a whole family of methods
* Centralised data structure that is easy for average users to understand
* Meta information regarding the methods in question (Authors, date, doi)
* Accessible data formats that are user readable with standard software suites

Issues REQUIRING resolution:

* The conflict between having safe, secure data, while allowing the user to retain the ability to alter values (as may occasionally be required)