smatrix
agroscience

# Data collection interface

1. Introduction

It is a common scientific method to perform **trials** (also called experiments) to gain empirical evidence. In agroscience this is done e.g., to evaluate fertilizer, pesticides/herbicides, or in the context of breeding.

Performing trials requires management of the whole process like planning, collecting data and finally an evaluation of the trial. Often a single trial management software (**TMS**) is used to support most of that. However, the actual data collection is usually done at a remote location like a field or a green house, creating the need for mobile data collection solutions.

This interface provides a simple communication mechanism between a trial management software and a data collection software (**DCS**). It can be used in both directions to

- Transfer the information required for data collection (trial structure) to the DCS,
- Return the trial structure enriched by the collected data to the TMS, where the data is
  - stored in a defined format and
  - uniquely mapped to the trial structure.

Trial management software (TMS) → Trial structure → interface → Data collection software (DCS)

Data collection software (DCS) → Trial structure + data → interface → Trial management software (TMS)
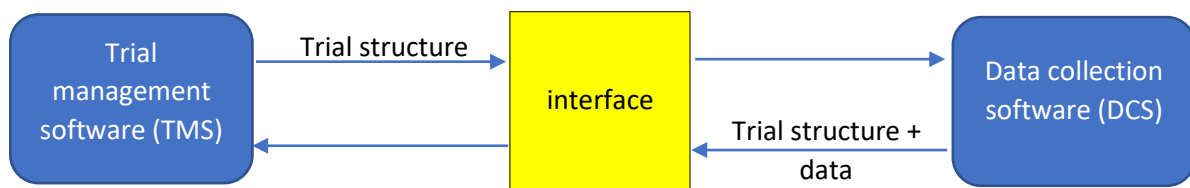
The interface is kept as generic as possible to cover different use cases in agroscience (or even other fields of work). For that purpose, the transferred information is abstracted.

1.1. Trial structure

Depending on the use case, the trial design can grow very complex. For the data collection however, the structure usually boils down to a simple table:

Trial unit number	Trait 1	Trait 2	Trait 3
1	data value	data value	data value
2	data value	data value	data value
3	data value	data value	data value
4	data value	data value	data value

Each row of this table stands for an object to assess, usually individual plants/trees or whole plots. Those are called **trial units** throughout the interface.

Each column of the table defines a criterion that needs to be assessed e.g., the height of a plant, a disease, or any other property. Those are called **traits** throughout the interface.
Note: Depending on your line of work, you might be used to other terms like assessment, characteristic, symptom, attribute, observation, or feature.

The table is spanned by a set of **trial units** and a set of **traits**. The goal is to collect a **data value** per trait/trial unit combination. In case of subsampling, it is also possible to collect multiple data values per trait/trial unit combination.

1.2.Introducing types

To keep the interface abstract, the meaning of individual **traits** and **trial units** is not transported. We rather focus on transferring the structure of the trial enriched by blocks of information – called **format** in the interface. Those contain a **format type** defining how they can be used. The format type consists of the following flags (multiple can be set to true):

- *is_identifying*
  The information is used by the data collection software. The information uniquely identifies a trait or trial unit and can be used
    - For display to the user
    - For voice control
        - for acoustic feedback to the user
        - for navigating within the trial by voice
  Important: Trial units and traits always need exactly one is_identifying information
- *is_pass_through*
  The information is required by the trial management software (TMS) for alignment. When receiving a trial, the TMS will use this information to identify traits and trial units and map them to their internal database. The data collection software ignores this information, unless one of the other types is set to true additionally.
- *is_info*
  The information might be used by the data collection software
    - for display as additional information for the surveyor
    - as criterion for error analysis

Note: All information is read-only and must remain unchanged when transferring the data back to the TMS!

2. Trial unit set

As mentioned above, a set of trial units needs to be defined. In the table, this corresponds to the definition of the rows.

2.1.Logical Representation

Before discussing the actual representation in the interface, we have a look at logical representation of the trial unit set.

2.1.1. Minimum use case

is_identifying is_pass_through
Plot
1
2
3
4

A single column can be used in the DCS (is_identifying) as well as for alignment in the TMS (is_pass_through). Consider the trial unit set as a list of plots

- o   We define a **level** (like a heading) e.g., plot
- o   Each trial unit has a unique **id-value** (i.e., the plot number) for that level

2.1.2. Trial units with multiple levels

is_identifying is_pass_through	
Plot	Plant
1	1
1	2
2	1
2	2

Trial units might not be identifiable uniquely by a single column. E.g., consider a trial where a trial unit is not a plot but a plant within a plot. In this case, we need to transfer the knowledge that the combination of these two columns is required.

- o   We define two **levels** e.g., plot and plant
- o   Each trial unit has two **id-values**, one per **level**. The combination of those is unique.

Note: maximum 3 levels are supported

2.1.3. Separate information for TMS/DCS

is_identifying		is_pass_through
Plot	Plant	TrialUnitId
1	1	X43951
1	2	X34231
2	1	X02394
2	2	X12928

The TMS might use a single column for alignment (e.g., an internal number). This however does not necessarily make sense for person collecting the data – he/she would require the location (plot and plant) shown in 2.1.2. Thus, there are two parallel blocks of information:

- o One for is_identifying with two levels as above.
- o One for is_pass_through with a single level.

2.1.4. Additional information per trial unit

is_identifying		is_pass_through	is_info	is_info
Plot	Plant	TrialUnitId	Treatment	Genotype
1	1	X43951	5%	A
1	2	X34231	10%	B
2	1	X02394	10%	A
2	2	X12928	5 %	B

Extending2.1.3, it might be useful to provide additional information on individual trial units. Such information can be passed using the type is_info.

Note: Those blocks are not essential for the trial structure and are therefor optional.

2.2. Representation in json (trial units)

The interface exchange format is based on json. To show the mapping of the above discussed logical representation to json, we consider a simple trial unit set with only a single trial unit:

is_identifying is_pass_through	format with format_type
Plot	Level with a code
1	trial_unit with id_value(s) – one per level

The highlighting of the code uses the same colors as the above examples. To align the id-values to the format & level, indices are used as shown below.

```
{
  "trial_unit_set_idx": 0,
  "formats": [
    {
      "format_idx": 0,
      "format_type": {
        "is_identifying": true,
        "is_pass_through": true,
        "is_info": false
      },
      "levels": [
        {
          "level_idx": 0,
          "code": "plot"
        }
      ]
    }
  ],
  "trial_units": [
    {
      "trial_unit_idx": 0,
      "id_values": [
        {
          "format_idx": 0,
          "level_idx": 0,
          "value": "1"
        }
      ]
    }
  ]
}
```

For the above discussed examples, please find the full json definitions in the following files:
2.1.1.json 2.1.2.json 2.1.3.json 2.1.4.json

2.3. Requirements

The following information is mandatory for a trial unit set:

- Exactly one format has "is_identifying": true.
  - This is required by the DCS.
  - It must have 1 to 3 levels.
    - The **"code"** in each level must be meaningful for surveyor (e.g., a word – not just plain numbers or internal ids). It is used to look up language specific representations, if provided. (later section)
- The id-values referring to the levels of the is_identifying format fulfill the requirements:
  - May not be empty
  - The set of id-value (referring to the is_identifying format) must be unique per trial unit.

2.4. Multiple trial unit sets

To allow more complex trials (and re-use of trial unit sets), the definition of trial unit sets contain a `trial_unit_set_idx` and are stored in an array named `trial_unit_sets`. Details on the useage are described in chapter 6.

3. Trait set

In addition to a trial unit set, we also need to define a trait set. In the table, this corresponds to the definition of the columns.

3.1. Logical representation per trait

For the traits we have a look at logical representation before discussing the actual representation in the interface.

3.1.1. Minimum use case

A single field can be used during DCS (is_identifying) and TMS (is_pass_through). The logic is the same as for the trial units. We use a single format and level. The code e.g., LeafRust can be used to look up language dependent versions (if defined).

is_identifying, is_pass_through	LeafRust

3.1.2. Trait is uniquely defined by a combination of multiple fields

If a single level is not sufficient, we can use up to 3 levels. That might be required e.g., to assess LeafRust on multiple leaf levels, or pest severity for different pests.

is_identifying, is_pass_through	LeafRust
	F-1

3.1.3. Additional trait information

To add additional information for each trait, further formats can be added as is_info. Opposite to trial units, where the level contains a code and the id-values are contained in each trial unit, traits contain both: the code and the value.

is_identifying, is_pass_through	LeafRust	
	F-1	
Is_info	unit	cm
Is_info	latin name	Puccinia triticina

3.2. Requirements per trait

For each trait, we require the following:

- Exactly one format needs "is_identifying": true.
  - It must have 1 to 3 levels.
    - The code in each level must be meaningful for surveyor (e.g., an EPPO-code or a word, no plain numbers, or internal ids). It is used to look up language specific representations, if provided. (later section)

3.3. Value related information per trait

In addition to its formats/levels, each trait also requires information on the data values that need to be collected.

- A unique index to map data-values to the trait.
  - trait_idx (int)
- The count of (sub)samples to be collected per trial unit (default 1)
  - subsample_count (int)

- The range of supported values (for validation & speech). This is just an index, used as reference to the actual definition of the value range. The complete definition of the value ranges is explained in chapter 5.
    - value_range_idx (int)

3.4. Representation in json (trait)

To show the mapping of the above discussed logical representation to json, we consider the following trait:

is_identifying, is_pass_through	LeafRust	
Is_info	latin name	Puccinia triticina
format with format_type	Level with code	value

The highlighting of the code uses the same colors as the above examples.

```json
{
  "trait_idx": 0,
  "subsample_count": 1,
  "value_range_idx": 0,
  "formats": [
    {
      "format_idx": 0,
      "format_type": {
        "is_identifying": true,
        "is_pass_through": true,
        "is_info": false
      },
      "levels": [
        {
          "level_idx": 0,
          "code": "LeafRust"
        }
      ]
    },
    {
      "format_idx": 1,
      "format_type": {
        "is_identifying": false,
        "is_pass_through": false,
        "is_info": true
      },
      "levels": [
        {
          "level_idx": 0,
          "code": "latin name",
          "value": "Puccinia triticina"
        }
      ]
    }
  ]
}
```

3.5. Multiple trait sets

To allow more complex trials (and re-use of trait sets), the definition of trait sets contain a trait_set_idx, and a list of traits. Multiple trait sets are stored in an array named trait_sets. Details on the useage are described in chapters 6 and 9.

Examples for trait sets with only a single trait as discussed in 3.1:

3.1.1.json 3.1.2.json 3.1.3.json

4. Data-values

During trial execution, each pair of trial unit and trait will receive one data-value (in case of subsampling several data-values).

4.1. Alignment

Data-values must be uniquely mapped to a trait and a trial unit. This is done using the corresponding indices. In case of subsampling, an additional index is used to identify the subsample.

- `"trial_unit_idx"` (number)
  - Identifies the trial unit this value belongs to.
  - Special case: Might be set to -1 for data values per execution, see 9.1
- `"trait_idx"` (number)
  - Identifies the trait this value belongs to.
  - Important: First the trial_unit_idx needs to be evaluated, as the trait_idx might refer to another trait set (see 9.1)!
- `"subsample_idx"` (number)
  - Indicates the subsample number
  - the value is optional, defaulting to 0
- `"value"` (string)
  - The actual collected value, validated (4.2) and written in a defined format (0)

4.2. Validation of the value

The content of `value` in a data value must be within the value range of the trait (see section 5).

The only exception to this rule is the **"unavailable"** value, which can be set independent of the value range.

4.3. Formatting of the value

To avoid ambiguities in the interpretation of values, the content of the string variable `value` is set in a defined way. It does not depend on language or locale.

- Values are always transferred as string.
- The representation of the "unavailable" value is part of the value_configuration, see chapter 7.
- Other values follow the formatting defined in the corresponding value range. That is, the value range pointed out by the trait this data value belongs to.

4.4. Comments

Comments are aligned in the same way as data values, see section 4.1.
The only differences are:

- They are free text don't have limitations of a value range.
- They are always considered optional. Because they are free text, they can't be reliably interpreted in TMS.
- To differentiate them from data values, that are interpreted in TMS, they are stored in a separate array "comments" per execution.

Example:

```
"comments" :
[
  {
    "trial_unit_idx" : 0,
    "trait_idx" : 0,
    "subsample_idx" : 0,
    "value" : "Comment for the first subsample of the first trait
and trial unit of an execution"
  }
]
```

5. Value Ranges

A value range defines which values can be written to data-values. The **"unavailable"** value is always supported, independently from the defined value range.

For the whole trial, several value ranges can be defined in the `"value_ranges"` array. Each trait specifies the supported value range using the value_range_idx (see 3.3). Note that the same value range can be used by multiple traits.

There are several possibilities to define a value range. Note however, that only one of these may be used per value range!

5.1. Numerical value range

To define a range of numbers, a number generator can be used. It has the following parameters:

- **"from"** (number): The first number in the range
  - Needs to be greater or equal to 0.
  - Is included in the range.
- **"to"** (number): The last number in the range
  - Needs to be greater than the value in **"from"**.
  - Is always included in the range (regardless of **"by"**)
- **"by"** (number): The step size used when generating numbers from **"from"** to the **"to"** value.
  - It is optional (defaults to 1)
  - Must be 0.001 or larger (maximum 3 decimals are supported)
- **"decimal_separator"** (string): The string to be used as decimal separator
  - Optional (defaults to ".")
- **"group_separator"** (string): The string to separate the groups (e.g. thousands)
  - Optional (defaults to an empty string)
  - Has to be different from "decimal_separator"

Example for a value range, allowing the numbers from 0 to 9 with 2 decimals.
E.g., 0, 0.01, 0.02, … 8.98, 8.99 and 9 are valid values:

```
{
    "value_range_idx": 0,
    "number_generators":
    [
      {
        "from": 0,
        "to": 9,
        "by": 0.01
      }
    ]
}
```

Limitation: Even though `number_generators` **is foreseen as array, current only one number generator is supported!**

5.2. Date

A `date_generator` can be defined to support dates. It has only a single parameter, defining its format:

- **"format"**: (string) Defines how the date will be stored/displayed.
  - Possible values (case insensitive) are: `DayOfYear`, `Default`, `yyyy-MM-dd`, `MM/dd/yyyy`, `dd/MM/yyyy`, `dd.MM.yyyy`, `d.m.yyyy`, `dd-mm-yyyy`, `yyyyMMdd`, `yyyy/MM/dd`
  - **"default"** is the same as **"yyyy-MM-dd"** (ISO 8601)

Example for a value range, allowing dates. The data is stored in the default format:

```
{
    "value_range_idx": 1,
    "date_generator":
    {
        "format"="default"
    }
}
```

5.3. Value-List

Instead of the above generators you can define the individual supported values in a list. Each entry is defined by the following parameters:

- **"code"** (string)
  - The content is used to look up language specific representations for display, speech recognition and speech feedback.
  - If no **"storage"** is specified, it also used in data values.
- **"storage"** (string)
  - The content is used in data values.
  - Is optional. If not specified, the value in **"code"** is used in data values.

An example could be a value list with colors:

```
    {
        "value_range_idx": 2,
        "value_list": [
            { "code": "yellow", "storage": "1" },
            { "code": "green", "storage": "2" },
            { "code": "brown", "storage": "3" },
            { "code": "dark brown", "storage": "4" }
        ]
    }
```

Or, if without different storage values using the codes only:

```
{
    "value_range_idx": 2,
    "value_list" : [
        { "code": "yellow" },
        { "code": "green" },
        { "code": "brown" },
        { "code": "dark brown" }
    ]
}
```

Note: It is possible to have entries with some other entries without "storage" defined in the same value range

5.4. Free Text

If the above definitions are not sufficient for the value range definition, free input text can be allowed:

- "allow_free_text" (boolean)
    - If set to true, any input string can be written to the data values.
    - Optional flag defaults to false.
    - **Important: It is highly recommended not to use this setting, because:**
        - **The input cannot be validated during data collection.**
        - **Automatic evaluation of free text in the TMS is not possible.**
        - **Only usecase for this are "traits per execution", see 9.2**

6. Executions

For many trials, data collection is not only required once, but at several points in time – called **execution**s in the interface. Each of these executions requires its own table discussed above.

6.1. Logical representation

So far, we have discussed trial unit sets (defining rows of a table) and trait sets (defining the columns of a table). We even allow for multiple trial unit sets (section 2.4) and trait sets (section 3.5). The missing part is to define which trial unit set shall be used together with which trait set to span a table.

This is defined per execution, allowing re-use of the same e.g., trial unit set in different executions.

6.2. Executions in json

The interface contains an "executions" array, with at least one execution within. An execution has the following parameters:

- `"execution_idx"` (number)
  - Matching the index in the array of executions (0-based)
- `"identifier"` (string)
  - Used by the TMS to identify the execution and align the data-values.
  - Required if trait sets are re-used across executions, otherwise optional.
- `"table"` (object)
  - `"trial_unit_set_idx"`: (number)
    - Refers to the trial unit set used in the table.
  - `"trait_set_idx"`: (number)
    - Refers to the trait set used in the table.
- `"execution_trait_set_idx"` (number)
  - Additional data collection is required as discussed in chapter 9
  - Optional. If not set (or -1), there is no additional data to collect.
  - If set, it refers to a trait set.
- `"data_values"` (array)
  - Contains the collected data-values as described in chapter 4 and section 9.1.
- `"comments"` (array)
  - Comments are additional texts, stored in the same format as data_values, see section 4.4.

Example with two executions:

```
"executions" : [
  {
    "execution_idx": 0,
    "identifier": "21312132131"
    "table":
    {
      "trial_unit_set_idx": 0,
      "trait_set_idx": 0
    },
    "execution_trait_set_idx":-1,
    "data_values": [],
    "comments": []
  },
  {
    "execution_idx":1,
    "identifier": "9812132133"
    "table":
    {
      "trial_unit_set_idx": 0,
      "trait_set_idx": 1
    },
    "execution_trait_set_idx":-1,
    "data_values": [],
    "comments": []
```

```
    }
]
```

7. Global information

- The version of this interface needs to be passed to ensure compatibility.
    - o **Important:** the interface is not yet released! current version is `"0.2.0"`
    - o in json that is contained as follows:
      `"interface_version": "0.2.0"`

- Information about the trial management software is optional but can be helpful for debugging. In addition, it could be displayed to the user for information where this trial was created.
    - o `"name"`: String
    - o `"version"`: string
    ```
    "trial_management_software":
    {
        "name": "name of the trial management software",
        "version": "1.0.1"
    }
    ```

- Information for trial identification:
    - o `"trial_identifier"`: string
        - ▪ Optional. This can be used TMS to identify the whole trial. This corresponds to the "is_pass_through" information for traits and trial units. It will not be used by the DCS.
    - o `"trial_display_name"`: string
        - ▪ Optional. This can be used by the DCS for display to the user. This corresponds to the "is_identifying" information for traits and trial units.
    - o Examples:
      ```
      "trial_identifier": "trial123Loc920",
      "trial_display_name": "herbicide trial A721"
      ```

- Optional readonly-information about the trial for display in the DCS. It can be used to pass information about the trial setup, the location of the field, contact persons or anything else, the user of DCS needs to know.
    - o `"trial_information"`: array of key/value pairs
        - ▪ Optional. each entry contains two parameters
            - • `"key"` : string
              A name to describe the information
            - • `"value"` : string
              Any required text
    - o Example:
      ```
      "trial_information" :
      [
          {
              "key" : "street address",
              "value" : "Neudorfer Strasse 123, Bonn, Germany"
          },
          {
      ```

```
            "key" : "contact person",
            "value" : "John Doe"
        }
    ]
```

- The value configuration impacts the data values. So far this is limited to a single definition, but might be expanded in the future, e.g., to enrich data values for automatic timestamps or geocoordinates.
  - o "n_a_representation": (string)
    - The content will be used in data_values when a "not available" value was recorded.
    - it can be any string
  - o It is optional. If not specified, it defaults to "---""max_number_subsamples" : (int)
    - Limit for traits added in DCS: The maximum allowed number of subsamples
    - Optional. Default value is 1
  - o "value_lists_supported" : (Boolean)
    - Limit for traits added in DCS
      - true: Value lists may be used for added traits
      - false: Value lists may not be used for added traits
      - Optional. Default value is true
  - o "date_format" : (string)
    - Format for date value ranges of traits added in DCS. Logic is identical to the "format" in 5.2
  - o "decimal_separator" : (string)
    - Format for number value ranges of traits added in DCS. Logic is identical to the "decimal_separator" in 5.1
  - o "group_separator"
    - Format for number value ranges of traits added in DCS. Logic is identical to the "group_separator" in 5.1

```
"value_configuration":
{
    "n_a_representation": "N/A",
    "max_number_subsamples" : 100,
    "value_lists_supported" : false,
    "date_format" : "dd.MM.yyyy",
    "decimal_separator" : ",",
    "group_separator" : "",
}
```

8. Representations / Pronunciations

All "`code`"s in the interface are language invariant strings. If they belong to a format used by the DCS (is_identifying or is_info set to true), they serve multiple purposes:

a) For display – "what the user will see."
b) For automatic speech recognition – "what the user can say."
c) For text-to-speech – "what the user will hear."

There are a couple of reasons, why the same string cannot serve all those purposes. In those cases, other representations are required or at least desired to improve the data collection:

- The code is an abbreviated word or a code (e.g., EPPO-code)
    - cannot serve b) and c)
    - requires a pronounceable representation
- The code is a regular e.g., English word, but the trial is implemented in another country collecting the data in another language.
    - cannot serve any of the above
    - requires localized and pronounceable representations
- The code is rather long, but display on mobile devices has limited size
    - could be improved using an abbreviated display representation a)

To overcome this, a dictionary can be passed in the json to provide representations for each code. Note that this is not mandatory but recommended if the information is available in the TMS.

8.1. Representations in json

A representation is defined by the following parameters:

- "culture" (string)
    - Defines the language where this representation applies.
    - It is a two-letter language code as defined in ISO 639-1
    e.g., "en" for English, "de" for German, "fr" for French, …
- "text" (string)
    - This is the string that should be used instead of the code.
- "for_display" (Boolean)
    - If true, the content of "text" shall be used for display a)
- "for_asr" (Boolean)
    - If true, the content of "text" shall be used for speech recognition b)
- "for_tts" (Boolean)
    - If true, the content of "text" shall be used for feedback c)

Example for a representation, that is usable in English for all purposes:

```
{
    "culture": "en",
    "text": "phytotoxicity",
    "for_display": true,
    "for_asr": true,
    "for_tts": true,
}
```

8.2. Dictionary entries in json

For each code, a dictionary entry can be defined by the following parameters:

- "code": (string)
  - The code the representations shall be used for
  - It is used as key when looking up representations.
- "representations": (array)
  - Contains several representations described in 8.1

Note that there are some requirements to which/ how many representations are supported. If a dictionary entry contains one or more representations for a language,

- exactly one representation must be defined as type `"for_tts": true`.
- exactly one representation must be defined as type `"for_display": true`.
- At least one representation must be defined as type `"for_asr": true`.

Example for a dictionary entry with representations for German and English.

```json
{
  "code": "PHY1PP",
  "representations":
   [
     {
        "culture": "de",
        "text": "Phytotoxizität",
        "for_display": true,
        "for_asr": true,
        "for_tts": true,
     },
     {
        "culture": "de",
        "text": "Phyto",
        "for_display": false,
        "for_asr": true,
        "for_tts": false,
     },
     {
        "culture": "en",
        "text": "phytotoxicity",
        "for_display": true,
        "for_asr": true,
        "for_tts": true,
     }
   ]
}
```

8.3. Dictionary in json

All required dictionary entries as defined in 8.2 are contained in a `"dictionary"` array.

9. Traits per execution

Some trials require collection of additional data values, that are outside the previously defined table. Examples could be development stages (e.g., BBCH) for the crop or a specific weed.

The data values are not collected per trial unit but refer to the whole trial at a specific execution. This can be defined as another trait set as discussed in chapter 3.

pest 1	pest 1	pest 1	pest 2	pest 2	pest 2	Crop	Crop	Crop
BBCH	BBCH	BBCH	BBCH	BBCH	BBCH	BBCH	BBCH	BBCH
minimum	majority	maximum	minimum	majority	maximum	minimum	majority	maximum
data value	data value	data value	data value	data value	data value	data value	data value	data value

To configure such an additional collection of data values, the execution needs to specify the corresponding trait set index in `"execution_trait_set_idx"` (see chapter 6).

9.1. Data values

The data-values still need to be uniquely aligned to traits (and trial units if specified). The indices shown in 4.1 are sufficient, but must be evaluated together:

- If the `"trial_unit_idx"` is -1:
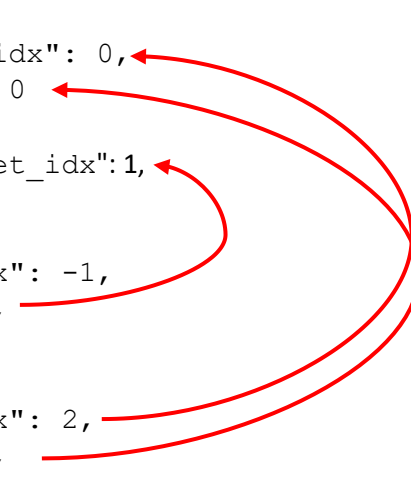
smatrix
agroscience

- o `trait_idx` refers to trait set defined by `"execution_trait_set_idx"`
- Otherwise, the indices refer to the sets in the `"table"` of the execution:
  - o `trial_unit_idx` refers to the trial unit set `"trial_unit_set_idx"`
  - o `trait_idx` refers to the trait set `"trait_set_idx"`

Example :

The value "80" refers to trait_idx 0 of trait set 1 (because trial_unit_idx is -1)

The value "90" belongs to trait_idx 3 of trait set 0 and trial_unit_idx 2 of trial unit set 0

```
{
    "execution_idx": 0,
    "identifier": "21312132131"
    "table":
    {
      "trial_unit_set_idx": 0,
      "trait_set_idx": 0
    },
    "execution_trait_set_idx": 1,
    "data_values": [
      {
        "trial_unit_idx": -1,
        "trait_idx": 0,
        "value": "80"
      }, {
        "trial_unit_idx": 2,
        "trait_idx": 3,
        "value": "90"
      }]}
```

9.2. Free text

For Traits per execution, "AllowFreeText" (see 5.4) is a viable option. E.g. to log the Name of the person executing the survey.

10. Summary – high level json layout

In the previous chapters we have discussed the individual parts of the interface. In summary, we can define the interface by the following:

- global information (chapter 7)
- trial unit sets (chapter 2)
- trait sets (chapter 3)
- executions (chapter 6)
- value ranges (chapter 5)
- dictionary (chapter 8)

A corresponding json consist (just looking at the highest level) of the following:

```
{
  "interface_version": "0.2.0"
  "trial_identifier": "",
  "trial_display_name": ""
  "trial_information": []
  "trial_management_software": {}
  "value_configuration": {}


  "trial_unit_sets": [],
  "trait_sets": [],
  "executions": [],
  "value_ranges": [],
  "dictionary":[],
}
```

11. Examples

- Minimum example
  - Trial units and traits are defined by a single format
  - Value range use number_generators

Plot	Aphids	Leaf Rust	Mildew
1	1	2	3
2	4	5	6
3	7	8	9

Example1.json Example1_with_data.json

- Minimum example with multiple executions
  - same as above, but with two executions. Here, both trial unit set and trait set are re-used, duplicating the table.
  - The first execution has the same data as above, the second table has the following data:

Plot	Aphids	Leaf Rust	Mildew
1	11	12	1
2	21	22	2
3	31	32	3

Example1_twoExecutions.json

- Complex example
  - Trial units are defined by two formats
    - one with two-levels for is_identifying
    - one with a single level for is_pass_through (using Guids)
  - Traits are defined by a single format
  - Value ranges use date_generator, value_list and number_generator
  - Representations are provided for codes in multiple languages

is_identifying	is_pass_through	is_pass_through is_identifying	is_pass_through is_identifying	is_pass_through is_identifying
Tray	Plant	RowId	Begin Of Flowering	Height	Phytotoxicity
1	1	a3f17d43-…	2021-04-21	small	1.6
1	2	b914e749-…	2021-04-22	medium	10.81
1	3	98f41faa-…	2021-04-23	high	7
2	1	3352b26b-…			
2	2	4cb9175b-…			
2	3	19953118-…			

Example2.json Example2_with_data.json