

Sysrevving

Gjalt-Jorn Peters

2022

Table of contents

Preface	5
1 Types of reviews	6
1.1 Narrative Reviews	6
1.2 Conceptual Reviews	6
1.3 Systematic Reviews	6
1.4 Meta-analyses	6
1.5 Scoping Reviews	7
I Planning	8
2 Introduction to Planning	9
2.1 Overview	9
2.2 Be explicit for redundancy, transparency, and future you	10
3 Research Questions	11
3.1 Systematic Reviews	11
3.2 Scoping Reviews	11
4 Planning the Synthesis	13
5 Planning the Extraction	14
5.1 Introduction to extraction	14
5.2 Entities	14
5.2.1 Identifiers and titles	14
5.2.2 Descriptions and extraction instructions	15
5.2.3 Values to be extracted	15
5.2.4 Hierarchical structure and container entities	15
5.2.5 Repeating entities	17
5.2.6 Clustering entities ('lists')	17
5.3 Value Templates	18
5.4 Details of the Rxs specification	19
5.4.1 The <code>entities</code> worksheet	20
5.4.2 The <code>valueTemplates</code> worksheet	21
5.4.3 The <code>definitions</code> worksheet	22

5.4.4	The <code>instructions</code> worksheet	22
5.4.5	The <code>texts</code> worksheet	22
5.5	Post-hoc entity specification: <code>Txs</code> specifications	22
6	The Categorization-Coding Continuum	23
6.1	Categorization	23
6.2	Ambiguity	24
6.3	The cost of categorization	24
6.4	Coding after extraction	25
6.5	The categorization-coding continuum	26
7	Planning the Screening	27
8	Planning the Search	28
8.1	Query Crafting	28
8.2	Queries as logical expressions	28
8.3	Database fields	29
8.4	Wildcard characters	29
8.5	Team Consensus and Expert Consultation	30
8.6	Databases versus Interfaces	30
8.7	“Smart” searching	31
8.8	Query validation	32
8.9	Exporting query hits	32
9	(Pre)registrations	33
II	Execution	34
10	Executing the Search	35
10.1	Running your queries	35
10.2	Exporting query hits	35
10.3	Importing your search hits	35
10.4	Deduplicating your search hits	35
11	Executing the Screening	36
12	Executing the Extraction	37
12.1	Data management	37
12.2	Extracting entities	37
12.3	Validation of extracted entities	37
12.4	Contact with authors	37
13	Executing the Synthesis	38

III References and Appendices	39
14 Example Projects	40
14.1 NITRO	40
14.1.1 Real-life projects	40
15 Glossary	41
16 Notes	44

Preface

The SysRevving book is an open access book about doing open and systematic systematic reviews using open source tools. That is not a typo: it really says systematic systematic reviews, reflecting how this book aims to make systematic reviews as systematic as possible. This is tied in with the aim of making them as open, transparent, and machine-readable as possible.

The scientific literature is rapidly expanding, and keeping up-to-date by reading new journal issues as they come out has been impossible for many years already. As a consequence, two practices are becoming increasingly common: PhD. candidates starting their project with a systematic review to get a hopefully unbiased overview of the literature in a field, and conducting living systematic reviews that can be updated with relatively little effort to provide an up-to-date database to answer specific questions.

At the same time, the workflow adopted in many systematic reviews is often partly designed anew for each systematic review. Decisions, procedures, and documentation are often only human-readable and scattered over multiple files, sometimes even in proprietary file formats or formats used by free but not free/libre open source software (FLOSS). This book aims to support a systematic workflow geared towards optimizing transparency and machine-readability.

The idea is that optimizing transparency and machine-readability means that systematic reviews can ultimately become more scalable, interoperable, and extensible. This in turn means that there is less wasted effort and eventually, systematic reviews can be accelerated ('revved up', so to speak).

If you prefer, you can download [the PDF version](#) or [the Ebook version \(.epub\)](#) of this book.

1 Types of reviews

Scoping reviews, meta-analyses, systematic reviews, narrative reviews, conceptual reviews.

1.1 Narrative Reviews

1.2 Conceptual Reviews

1.3 Systematic Reviews

Systematic reviews differ from conceptual or narrative reviews in that they can afford the right to make strong statements by virtue of the employed procedures: those can and should be extremely rigorous, systematic, transparent, and reproducible.

This means that as much as possible, the decisions that are taken must be clearly documented and justified, so that the inevitable biases that the research team bring to the project can be taken into account when interpreting the results. It also means that the preparation phase is vital.

1.4 Meta-analyses

Meta-analysis is a homonym, referring simultaneously to a statistical approach and to a class of systematic reviews. The statistical approach comprises techniques to quantitatively synthesize multiple estimates of the same population parameter. Some of these techniques are relatively simple (such as when synthesizing two or more correlation coefficients), and some are very sophisticated (such as when using multi-level meta-regression).

The class of systematic reviews known as meta-analyses are those systematic reviews where the research question can be answered by quantitatively synthesizing a set of estimates, and where the heterogeneity of those estimates is sufficiently low to warrant such quantitative synthesis. Confusingly, the estimates of that heterogeneity require conducting a meta-analysis (the statistical approach). This means that in systematic reviews where the reviewers aim to quantitatively synthesize multiple estimates, but where those estimates turn out to exhibit so much heterogeneity so as to preclude a statistical meta-analysis, a statistical meta-analysis

is conducted nonetheless to obtain those heterogeneity estimates. The result is a systematic review that is not a meta-analysis but that does include a statistical meta-analysis.

In such situations, the reviewers have to synthesize the estimates in another way, often resorting to qualitative integrations or to using visualisations. An insightful visualisation can be a forest plot, a visualisation typically used by statistical meta-analyses to illustrate how the synthesized estimate compares to the estimates from the separate studies – but then omitting the synthesized estimate.

1.5 Scoping Reviews

Scoping reviews or evidence maps (depending on who you ask, these can be the same or slightly different) differ from most types of systematic reviews in that they don't answer substantive research questions (note that when I use "systematic reviews", that also includes meta-analyses). Instead, they provide an overview of the scope of the literature: in a sense, they can tell you which research questions *can* be answered with systematic reviews.

Where most systematic reviews synthesize the evidence itself, often aiming to provide a more conclusive answer to the same or similar research questions as the included primary studies asked, scoping reviews synthesize the metadata about that evidence. They can tell you things like when most studies were conducted; which (sub)topics received most attention when; which study designs were used and whether that was associated to (sub)topic; how studies were distributed geographically; which sample sizes were common; whether any of those variables shows trends over time; et cetera.

Scoping reviews also produce an extensive database of literature, and are an excellent starting point for focused systematic reviews. Those also become much easier to plan, since you'll know how many studies are available. Depending on the comprehensiveness of your scoping review's extraction, you may even be able to skip the search and screening phases of those systematic reviews, since you already know which articles to include. Especially in combination with a decentralized approach to extraction, this means that scoping reviews can enable very efficient mapping of the literature.

Part I
Planning

2 Introduction to Planning

2.1 Overview

Planning a systematic review works in the opposite order of conducting it. Specifically, when planning, you and your research team have to achieve consensus on the following matters, in this order:

- 1) The goals and/or research question(s);
- 2) As a function of this, the entities you will extract (see the [Planning: Extraction chapter](#), chapter @ref(planning-extraction) in this version of the book);
- 3) As a function of these entities and the goal/research questions, the exclusion criteria you will use during screening;
- 4) As a function of the goal/research questions and the exclusion criteria, your search strategy, which includes:
 - 1) the conceptual form of the query you will use;
 - 2) which database(s) and interface(s) you will use;
 - 3) the conceptual query translate to each database/interface combination;
 - 4) additional strategies, e.g. forward and backward citation searches;

This will determine the scope of your review. Although these steps depend on the previous steps' output, in practice, this process is often nonlinear and iterative. For example, you often test draft queries in your interface/database combinations to see how many hits you obtain, potentially deciding to adjust your exclusion criteria or even your goals or research questions depending on what you find to ensure that the systematic review stays within the scope determined by your resources (time, funding).

2.2 Be explicit for redundancy, transparency, and future you

To err is human, and therefore, in scientific endeavors, it is best to never count on a single human not erring. The solution to this is two-fold. First, don't let humans perform tasks that computers can perform; and second, implement redundancy. That means that for tasks that have to be performed by humans, always have at least two people perform every task independently and check for consistency in the results.

If you can afford such redundancy, there is a clear penalty for sloppy planning. If your definitions, descriptions, and instructions leave room for interpretation, the laws of probability decree that that room will inevitably be taken sooner or later. This will manifest as heterogeneous results that will be labor-intensive to reconcile. In a non-trivial proportion of cases, such divergent results may prove almost impossible to reconcile, as they may bring to light fundamental problems with the tasks, definitions, and procedures you specified during your planning.

However, in many cases, full redundancy is not practically attainable. For example, when a systematic review is conducted in the context of a bachelor's thesis, a master's thesis, or a PhD. thesis, only one screener and one extractor may be available. Similarly, in many projects, having multiple independent synthesists is not feasible. In such cases, sloppy planning and sloppy documentation of the planning is not penalized as explicitly and as acutely.

Therefore, in such cases, it is particularly important to pay special attention to clearly documenting your plans, definitions, decisions, and their justifications. In addition, it is important to not forego developing instructions for each task as if you were not the only person who will conduct them. Although redundancy may not require this, there are two reasons to do this nonetheless.

First, as the transition towards open science has shown, there is much to gain from exercising transparency in science, both epistemologically and operationally. Epistemological benefits include error-detection, easy and accurate identification of risks of bias, and availability of process information for meta-scientific interrogation. Operational benefits include facilitating learning from other researchers and prompting more elaboration through the awareness that one works in public.

Second, systematic reviews tend to be both very valuable and take a lot of time, and the process tends to be very similar every time. These characteristics mean that you will likely come back to your earlier documentation and plans, either to remember what exactly you did and why, or because you want to adapt and re-use some elements of your process. You will save Future You a lot of time and effort by exercising some minimal hygiene throughout your project in terms of data management and clear documentation, including instructions that may seem unnecessary at the time.

3 Research Questions

The research questions guide much of the planning (and so, execution) of a systematic review. I say “much” because when you’re planning a systematic review and it’s not necessarily a one-shot endeavour, you will often want to anticipate as many future needs and wishes as possible, which extends your planning beyond your current research questions.

Still, any systematic review will be conducted with a specific initial goal in mind, and because that goal will often be obtaining one or more answers, the questions to be answered are a useful way to structure the planning.

These research questions will always contain one or more concepts. Each of these concepts will relate to one or more [entities to extract](#) (see below). Once you have decided on your research question, therefore, you can decide on the entities to extract.

However, in practice this process is nonlinear and iterative. Any given research question (or more accurately, any given set of entities to extract) implicitly determines the scope of the review, because the exclusion criteria are based on the research questions and the entities to extract, and the search strategy (e.g. the query) is based on the research questions, the entities to extract, and the exclusion criteria. As such, there is always some correspondence between the research question and the number of sources that your search strategy will yield (and that will have to be screened).

Therefore, you will usually develop all of these in parallel. For example, it is common to test different queries until the number of hits is feasible given the available resources. Depending on your screening capacity, you may be forced to revise and limit the scope of the research question(s). Similarly, if your query yields very few hits, you may want or need to broaden it (and therefore, broaden your research question(s)) to eventually obtain worthwhile results.

3.1 Systematic Reviews

3.2 Scoping Reviews

Research questions in scoping reviews ask what researchers did. They can concern anything from whether different geographical regions prioritize different topics, whether sample sizes increased or decreased over time, which definitions researchers use, and which measurement instruments researchers use, via things like which study designs are used to answer which types

of questions and how paradigms change over time to whether shifts occurred in researchers' underlying philosophy of science or epistemological perspectives.

4 Planning the Synthesis

Where in primary research, the term used for the process by which one arrives at answers to a research question based on the collected data, in a systematic review, this process is called synthesis. This nicely captures the aim to combine information from multiple sources to yield a (hopefully coherent) overview.

Synthesis of systematic review results shares many characteristics with analyses in primary research. In both cases, the process consists of dozens to thousands of subjective decisions; in both cases, the process can be challenging and complicated; and in both cases, the way the process is challenging and complicated depends on the type of data being processed. For systematic reviews, syntheses tends to be easier as sources are more similar.

For example, a relatively simple synthesis could be a meta-analysis of studies that all have the same design and used the same measures for the same variables, for example when aggregating randomized controlled trials for a specific COVID-19 vaccination. In such scenarios the studies often share the same ontological and epistemological perspectives, and a single effect size estimate can often be extracted from each source, all in the same metric. In addition, variation in extracted effect sizes most likely reflects sampling variability and contextual factors. Both can be statistically modeled (assuming those contextual factors were extracted).

An example of a relatively complicated synthesis is a systematic review that provides an integrative overview of a topic over multiple study types, for example everything that is known about why people

5 Planning the Extraction

5.1 Introduction to extraction

The extraction is the stage where the “data” are extracted from the identified [sources](#). This means that the information from or about the included sources has to be stored in an extraction script file.

In a systematic review, the extraction and synthesis stages are the hardest (unless a meta-analysis is possible, in which case the extraction stage is the hardest). This is because the information you want to extract will often be ambiguous, and sometimes it will not be available at all. This ubiquitous ambiguity means that the task of extracting information is typically not a matter of copying information over: instead it’s more like playing detective.

5.2 Entities

Planning the extraction is also the hardest part of planning a systematic review. Planning the extraction means specifying the R extraction script, which requires specifying the entities to extract, how they are hierarchically organized, and which value templates each entity uses.

An entity is anything that has to be extracted from a source, such as the year something was published, its authors, a definition that was used in a source, a theory that was studied, a study design, a sample size, a measurement instrument, the literal text of measurement instrument items, expressions from interview participants, effect sizes, or things like the number or figures, tables or words in a source.

During the planning phase, you decide which entities you want to extract and how. A garden-variety entity represents one type of thing you want to extract from your included sources. You will define them in the Rxs specification, a spreadsheet (see below for details).

5.2.1 Identifiers and titles

For every entity, you will choose an identifier, a title, and a description. The identifier is a unique machine-readable name for your entity. This will allow you to easily select all extracted data for a given entity during the synthesis phase. Identifiers can only contain lower- and uppercase Latin letters, Arabic numerals, and underscores, and must always start with a

letter. ¹ Titles are the human-readable equivalent, so basically just the name of the entity, without any constraints as to which characters you're allowed to use, but short.

5.2.2 Descriptions and extraction instructions

Descriptions are longer, and should contain at least two, and preferably three elements. First, the description should describe and define the entity. Since what you include here will be all that extractors, other researchers and interested parties, and Future You will have to go on, it pays to make an effort to be as explicit as possible.

Second, the description should contain explicit extraction instructions for the extractors — even if you yourself are going to be the only extractor (see the [Be explicit for redundancy, transparency, and future you section](#), section @ref(planning-intro-explicity) in this version of the book).

Third, ideally, the description should explicitly list one or more edge cases. Edge cases are examples of something that a source might contain where it is not obvious how it should be extracted correctly. By listing these and explicitly describing why that example should be extracted the way you specify, you help extractors (including Future You) understand better how you delineate your entity definition.

5.2.3 Values to be extracted

To specify which types of values can be extracted for each entity, value templates can be specified (see the Value Templates section below). For each entity, the valid values, default value, and examples specified in the value template can be overridden in the entity specification.

5.2.4 Hierarchical structure and container entities

Because the number of entities extracted from the sources in a systematic review can become quite large, and are often clustered together, entities have a hierarchical data structure. This means they form a tree: each source is a root that can have leafs and/or branches attached. In data tree terminology, terms from two vocabularies are mixed: tree terms, such as root, branch, and leaf; and family terms, such as parent, child, descendants, and ancestors.

To familiarize yourself with these terms, consider the tree in [?@fig-planning-extraction-tree-illustration](#).

¹If you're already familiar with regular expression, the regular expression is `[a-zA-Z][a-zA-Z0-9_]*`. If you're not already familiar with regular expressions: they're an extremely powerful tool to describe, search for, and replace text fragments, well worth at least a brief acquaintance.

```
# This {dot} chunk can't run yet as Chromium isn't installed on the server yet
///| label: fig-planning-extraction-tree-illustration
///| fig-cap: An example data tree with entities.
```

```
digraph treeIllustration {

    graph [rankdir=TB];
    node [shape=box,fontname=arial];
    edge [arrowhead=none];

    source -> general;
    source -> methods;
    source -> results;

    general -> publicationYear;
    general -> sourceAuthors;
    general -> sourceTitle;

    methods -> sample;
    methods -> method;
    methods -> variables;

    sample -> sampleSize;
    sample -> samplingStrategy;

    variables -> variable;

    variable -> variableIdentifier;
    variable -> measurementLevel;

    results -> associations;
    associations -> association;

    association -> associationIdentifier;
    association -> varId1;
    association -> varId2;
    association -> r;
    association -> t;

}
```

In this tree, the `source` itself is the root where all entities are attached. The three *container entities* attached to the root are `general`, `methods`, and `results`. These container entities are

used to organise other entities: nothing is extracted for those contained entities themselves, they just function to organise and represent their children. The children of the **general** container entity (**publicationYear**, **authors**, and **title**) are themselves leaves: they have no further descendants.

Of the entities specified as children of the **methods** container entity, the **method** entity doesn't have descendants either: that entity is also a leaf. The **sample** entity does have two children (i.e. is a branch), **sampleSize** and **samplingStrategy**, each leaves themselves (i.e. without children). The **variables** entity has one child that is itself a branch (**variable**), which has two children: **variableIdentifier** and **measurementLevel** (both leaves).

Finally, the **results** container entity contains one container entity called **associations**, which contains another container entity (i.e. a branch) called **association**, which contains three regular entities (i.e. leaves): **varId**, **r**, and **t**.

The position of an entity or container entity in the Rxs tree is specified by its parent, where the entity identifier of the parent container entity is specified.

5.2.5 Repeating entities

Sometimes, an entity, entity container, or clustering entity (see the next section) can potentially be extracted multiple times. For example, a source may report on multiple samples or may report multiple effect sizes. Therefore, some entities are repeating entities, which means the extraction script will be set up such that the corresponding lines can be copy-pasted multiple times. Normally, only clustering entities will be repeating. This will be explained in more detail in the next section.

5.2.6 Clustering entities ('lists')

In addition to container entities, that themselves contain no extracted data but are used to organize other entities, there are *clustering entities*. You can consider clustering entities as a special type of container entity that only contains leaf entities that are closely related to each other. In the extraction script template, these *clustered entities* (i.e. the leaf entities in a clustering entity) are placed on successive lines, with their titles, descriptions, value template descriptions, and examples all concatenated in one line after the bit where the entity itself is extracted.

There are two benefits to using clustering entities. First, they are more efficient during extraction, especially if the clustering entity is a repeating entity (see below). Second, **metabefor** has functions to supplement a clustering entity with entities from elsewhere in the extraction tree.

To illustrate this, again look at Figure @ref(fig:planning0extraction-treeIllustration). In this Rxs tree, there are two repeating clustering entities: **variable** and **association**.

The `variable` clustering entity contains two clustered entities: `variableIdentifier` (a unique identifier for each extracted variable) and `measurementLevel` (the measurement level of this variable). The `association` clustering entity contains five clustered entities: `associationIdentifier` (a unique identifier for each extracted association), `varId1` (the identifier of the first variable, referring to a `variable` clustering entity by its `variableIdentifier`), `varId2` (the identifier of the second variable, also referring to a `variable` clustering entity by its `variableIdentifier`), `r` (a Pearson correlation coefficient), and `t` (a Student t value).

Both the `variable` and `association` clustering entities are repeating entities. This means that they can each be extracted multiple times by copy-pasting the relevant lines in the extraction script. Because each clustering entity has a unique identifier, they can be referred to, and each `association` clustering entity refers to two `variable` clustering entities.

Now, imagine a systematic review on gym membership, exercise, diet, and BMI. The extractor might encounter a source where they extract four effect sizes in four `association` clustering entities:

- the Pearson correlation between height and weight;
- the Pearson correlation between weight and daily energy ingestion;
- the Pearson correlation between weight and daily exercise; and
- the Student t value for the association between gym membership and daily exercise).

The extractor also specifies the measurement level for each variable in five `variable` clustering entities.

During synthesis, `metabefor` allows supplementing the `association` clustering entities with the information specified in the `variable` clustering entities using the unique identifiers specified in `varId` and `varId2` and then looking for the corresponding clustering entities with that identifier in their `variableIdentifier` entity.

This is a trivially simple example, but this functionality is very powerful to extract efficiently and with high fidelity, while retaining flexibility and easily recombining the extracted entities during the synthesis stage to ultimately obtain data frames that lend themselves well to the intended synthesis.

5.3 Value Templates

Value templates are an efficient method to define a type of data to be extracted. The example `metabefor` Rxs specifications contain a number of common value templates:

- `numeric`: Any valid number
- `numeric.multi`: A vector of valid numbers
- `integer`: Any valid whole number

- `integer.multi`: A vector of integers (i.e. one or more whole numbers)
- `integer.length4.multi`: A numeric vector of years
- `string`: A single character value
- `string.multi`: A character vector (i.e. one or more strings)
- `countrycode`: A character vector of the ISO 3166-1 alpha-2 country code(s)
- `categorical`: A string that has to exactly match one of the values specified in the “values” column of the Coding sheet
- `generalPresence`: Whether the thing being coded was present or not.
- `string.mandatory`: A single character value that cannot be omitted
- `string.entityRef.mandatory`: A string that specifies another entity and which **MUST** be provided
- `string.entityRef.optional`: A string that specifies another entity (can be missing, i.e. NA)
- `string.fieldRef.optional`: A string that specifies another field in another entity (can be missing, i.e. NA).
- `matrix.crosstab`: A table with frequencies; variable 1 in columns, variable 2 in rows; always work from absence/negative/less (left, top) to presence/positive/more (right, bottom)
- `string.identifier`: A single character value that is used as an identifier and so is always mandatory and can only contain a-z, A-Z, 0-9, and underscores, and must start with a letter.

Each value template specifies a unique identifier, a description, optionally the valid values that can be extracted, a default value to insert into the extraction script template, one or more examples, an R expression to validate the extracted value (which implements the descriptions in the list above), and an error to show if that validation fails.

5.4 Details of the Rxs specification

The entities are specified in a spreadsheet called an Rxs specification. Rxs stands for R Extraction Script, and they are the machine-readable files that data from sources are extracted into. They are in fact R Markdown files that can be rendered as-is, but that can also be imported using `metabefor`. These files are created by `metabefor` based on the Rxs specification.

An very minimal example of such as spreadsheet is available at https://docs.google.com/spreadsheets/d/1Ty38B5GJ6zzr7E3rC_vQNOMKe-uCvIuHs3c. A more extensive example is available at <https://docs.google.com/spreadsheets/d/13MUf8qL4Zmc5V6AOvjO1GWeFC14IaaSl2zUT-Kk9tQc>. See the `@ref(example-projects)` chapter in the Appendix for a list of example projects.

A spreadsheet holding an Rxs specification has at least the following worksheets:

- `entities`: The specifications of the entities to be extracted in the systematic review.

- **valueTemplates:** The value template specifications: an efficient way to specify ‘data types’ for entities.
- **definitions:** Definitions of concepts used in the systematic review.
- **instructions:** Instructions for the extractors.
- **texts:** Texts to override `metabefor`’s default texts.

These will now briefly be described.

5.4.1 The entities worksheet

The `entities` worksheet has the following columns:

- **title:** A short human-readable label for the entity (basically its name).
- **description:** A longer human-readable description of the entity. Together with the value template description, this will form the instruction for the extractors, so make sure to clearly describe what they should look for in the sources.
- **identifier:** A machine-readable identifier for this entity. This may only contain lower and upper case Latin letters (**a-z** and **A-Z**), underscores (**_**), and Arabic digits (**0-9**), and must start with a letter. This is used to refer to extracted entities in the results, or when cross-referencing entities (e.g. in the `parents` column).
- **valueTemplate:** The identifier of the `valueTemplate` to use (see the `valueTemplates` worksheet).
- **validValues:** Overrides the `validValues` specified in the specified ‘valueTemplate’.
- **default:** Overrides the `default` value specified in the specified ‘valueTemplate’.
- **examples:** Overrides the `examples` specified in the specified `valueTemplate`.
- **parent:** The entity’s parent entity: in the hierarchical tree of extracted entities, the parent is the entity that this entity will fall under. For example, in the `Rxs` specification for the the example tree shown above, the entities `samplingStrategy` and `sampleSize` each list `sample` in the `parent` column.
- **list:** If `list` is set to `TRUE`, that designates this entity as a *clustering entity*. That means that the entities it contains are *clustered entities* that are presented in the extraction script in a `list()`. This allows for more efficient extraction of the child entities. However, is also means that in the tree of extracted entities, these child entities (i.e. the clustered entities) cannot themselves have child entities. In other words, those child entities are all leafs on the tree.
- **repeating:** Set `repeating` to `TRUE` for entities that can be extracted multiple times. This is useful for, for example, effect sizes or other statistics, which can be extracted multiple times for a given source, but always have the same specifications.

These columns are also included, but contain functionality that is both quite advanced / abstract and not yet fully implemented in `metabefor`:

- **collapsing:** To be added.

- **recurring**: To be added.
- **recurring**: Set **recurring** to TRUE for entities that can recurse: that can contain themselves.
- **identifying**: Set to TRUE if this is entity if an identifier.
- **entityRef**: It is often useful to specify that extracted information relates to a specific entity (usually an entity container). In such cases, this column can be used to specify which entity is referred to. This is then used during validation to verify whether in the tree object, the value specified for this entity occurs as one of the values specified for the **entityRef** entities. For example, when conducting a meta-analysis, it is typically useful to extract the variables measured in a study as well as estimates for associations between those variables. Using **entityRef** entities, it is possible to extract the measurement instrument used for the relevant variables only once, and then refer back to those entities using the **entityRef** entity.
- **fieldRef**: [this is advanced functionality that still has to be implemented in **metabefor**] Sometimes, extracted information does not relate to another entity, but to one specific value for an entity specified in the **entityRef**. The **fieldRef** field allows you to specify the identifier of the entity within the entity referenced in the **entityRef** entity to which the parent entity pertains.
- **owner**: This entity's owner - specifying an owner signifies that all entities with that identifier must contain at least one entity with the current identifier.

5.4.2 The valueTemplates worksheet

- **identifier**: The unique identifier of this value template, used in the **entities** worksheet to specify that this value template should be applied to an entity. This must be a machine-readable identifier, and so may only contain lower and upper case Latin letters (a-z and A-Z), underscores (_), and Arabic digits (0-9), and must start with a letter.
- **description**: A description of this type of value. This will be shown in the Rxs template for every entity that this value template will be applied to. Specifically, extractors will see this description printed below those entities.
- **validValues**: Optionally, a list of valid values. Each value must be separated by double pipes (||). For example: "Unknown" || "Present" || "Absent" means that one or more of those three strings must be extracted.
- **default**: The default value inserted in the Rxs template.
- **examples**: Examples of extracted values.
- **validation**: An R expression to validate the extracted entity.
- **error**: An error message to show if the validation fails.

5.4.3 The definitions worksheet

Here, you can specify definitions that are important in your project. They will be included in the extractor instructions, together with the contents of the `instructions` worksheet. There are two columns:

- `term`: A term.
- `definitions`: The corresponding definition.

5.4.4 The instructions worksheet

Here, you can specify instructions for your coders. There are two columns:

- `heading`: A heading, which will be included as a heading in the instructions.
- `definitions`: The instructions that should be displayed below that heading.

5.4.5 The texts worksheet

This functionality has not been implemented yet, but it will allow overriding the default texts produced by `metabefor`.

- `textIdentifier`: A unique identifier for the text fragment.
- `content`: The text fragment that should be used.

5.5 Post-hoc entity specification: Txs specifications

Tabulated Extraction Sheet specifications

6 The Categorization-Coding Continuum

6.1 Categorization

For some entities, the potential values an extracted entity can take are knowable in advance in a given systematic review context. For example, when reviewing primary studies in humans or other animals, the sample size must be a positive integer (e.g. 1, 2, 3, ...); and publication year will usually have to be a positive integer, often of four digits. In other cases, it is clear that free text will be extracted, for example, when author names or source titles are extracted.

For many entities, however, it is less obvious how to operationalize them. When something is extracted as free text, often as many unique values will be extracted as there are sources. This means that synthesis (i.e., “analysis”, see below) first requires transformation of those values. A list of raw free text values cannot be synthesized: the strings of characters have no encoded meaning, and cannot be collapsed or summarized. Nothing can be calculated from a list of free text values; and if that list is used in a table, that table will have as many rows or columns as the list of free text values has different values. Especially in scoping reviews, where including hundreds of sources can be quite common, this often isn’t feasible.

In many cases, this problem can be avoided by having extractors categorize that information during extraction. For example, imagine a scoping review into qualitative research practices in a given field. One of the entities that will be extracted is how the researchers coded the data. In this case, an infinite number of coding approaches can be used. Many textbooks on qualitative research use some categorizations to organize these. For example, coding can be categorized as “inductive” versus “deductive” or as “open” versus “axial”. Like any categorization, these simplify reality, making it easier to deal with for humans. If this simplification is not problematic given the scoping reviewers’ research question(s), they can choose to adopt one of those categorizations.

In that case, they would decide which categories to use (e.g. **inductive coding** and **deductive coding**, or two symbols representing these two categories, such as 1 and 2) and specify clear coding instructions for each (often with special attention to edge cases). After extraction, instead of having one or several sentences of free text extracted for each source (where the original authors describe their coding approach), they would then have a list with only possible two values (e.g. **inductive coding** and **deductive coding**, or 1 and 2). This lends itself to easy synthesis: the percentage of sources using inductive coding could easily be obtained, and it would be possible to answer questions such as whether that percentage seems stable over time, or differs between subdomains, or by geographical area.

6.2 Ambiguity

However, the extractors would also encounter sources where the authors used both types of coding - and they would encounter sources where a coding approach would be used that could arguably belong in either (or neither) category. There are two strategies to try and prevent such problems.

The first is developing very, very comprehensive coding instructions. If the scoping reviewers have a clear idea of all potential coding approaches, discussing all edge cases extensively in the coding instructions can ensure unequivocal (and correct) categorizations of most potential descriptions extractors can encounter in the sources. For example, the coding instructions can instruct extractors to categorize all sources using both inductive and deductive coding as “inductive” (or “deductive”, depending on what makes sense given the scoping review’s goals).

The second is putting a lot of thought into the categories that are used for each entity. For example, instead of using two categories, the scoping reviewers could add a third category **inductive and deductive coding**. They could also split the entity into two dichotomous entities, having extractors extract whether inductive coding was used into one, and whether deductive coding was used into another. By adding a third category **unclear** to each entity, ambiguous cases could be easily spotted - however, at the cost of no longer knowing what the extractor would guess if forced. That could be solved by adding more categories, for example extracting the entity **inductive coding** into categories **no**, **unlikely**, **likely**, and **yes**; or, alternatively, by adding a second entity that holds the extractor’s confidence in the categorization.

6.3 The cost of categorization

Each of these solutions to the problems caused by reality (including researchers’ decisions as extracted in scoping reviews) usually not being neatly organized into categories entail some costs. The more entities that are used to store the information extracted from the sources, and the more categories that are used for each entity, the less information is lost during extraction – but the more time and effort the extraction costs.

In addition, any categorization by definition means that what can be learned from the systematic review is limited to the “potential answer space” formed by what the systematic reviewers knew a priori. If a research question is “which coding approaches are used”, and the entities that systematic reviewers extract into are **inductive coding** and **deductive coding** (both with categories **no**, **unlikely**, **likely**, and **yes**), then the synthesis can never result in conclusions about the proportion of sources where the researchers reported they used guinea pigs, neural networks, or magic crystals for coding, even if a sizeable proportion of the sources

reports those approaches. Each of these three types of coding approaches will instead be categorized as either inductive coding or deductive coding (or potentially both) – if the coding instructions are of sufficient quality, they will be categorized unequivocally and consistently, but still, a lot of information will be lost.

This can be problematic depending on the research questions. Often, what the systematic reviewers *do not* see coming a priori can be the most interesting. When the nature or scope of the “potential answer space” is not the thing of interest (i.e., the researchers are interested in where the set of included sources falls in that space), the costs of categorization can be zero or low. However, when it is not clear in advance how that space looks, researchers may not be able to afford categorization at extraction time. In that case, coding can happen after the extraction stage.

6.4 Coding after extraction

When coding after extraction, during extraction the only decision the extractors face is which fragments to extract. They don’t need to interpret anything beyond identifying which part(s) of the source contain(s) the relevant information, which decreases the probability of errors. That interpretation then comes after.

The extracted original raw text fragments can then be exported to `.rock` files that can be coded using the Reproducible Open Coding Kit (ROCK) standard. The coded files can then be imported again and merged into the object holding all extracted data.

There are a lot of advantages to this approach. First, it makes the review much more transparent. It’s easy for others to see which fragments were selected, and so what the results were ultimately based on. Second, it lends itself well to rigorous quality control: having a file with extracted fragments coded by multiple coders is relatively straightforward and ‘cheap’ (timewise), since the selection of the relevant fragments is often a large part of the task. Third, it scales very well: the tasks of selecting the relevant fragments and the interpretation of those fragments can be distributed between multiple extractors and coders. Fourth, closely related, it enables a decentralized approach, where different groups can work on different parts of a project. This means that it enables involving students or citizen scientists. Fifth, it provides flexibility regarding effort distribution over time. If twenty entities are extracted as free raw text fragments, reviewers can decide to start with coding the first five, which might be enough to answer their main research questions. The other fragments can then be coded later on, without delaying the rest of the project. Sixth, it allows relatively efficient re-coding using different categories, which is for example very useful when conducting living reviews, where insights about how to categorize can change over time.

There are also disadvantages to this approach. First, it costs more time to record raw text fragments (which requires copy-pasting, usually from PDFs which also means it often also requires some cleaning of the pasted text) than it takes to record a selection from a predefined

set of categories. Second, experienced extractors develop competences that make them more efficient and more consistent over time. By cutting up the tasks and potentially distributing those over more people, this training effect decreases.

6.5 The categorization-coding continuum

Whether a given entity is extracted as a raw free text fragment or categorized during extraction, and if the latter, which categories are used and whether the entity is split up into multiple entities has to be decided in the planning stage. Changing this decision once the extraction has started is *extremely* expensive in terms of time, energy, and “error-prone-ness”, which means that it is worthwhile to put a lot of thought into this decision for every entity.

In fact, together with which entities are extracted, how to extract each entity is the most important decision taken when planning a systematic review. These decisions determine for a large part how much time and energy the review will take, as well as how flexible the compiled database will be, how extensible the review will be, and whether the process is scalable and lends itself to decentralization.

Whether an entity should be extracted as raw free text fragments that are then later coded, or categorized during extraction, or any of the options in between (e.g. categorization into one entity with a second entity to specify raw text fragments in case of a misfit with the prespecified categories; or coding into predefined categories but using multiple entities and many categories to lose as little information as possible), depends on a number of things. For example, when few resources are available (e.g. time, people), extracting raw text fragments and coding afterwards may not be feasible. If conversely, if the reviewers aren’t confident they can specify a well-defined set of mutually exclusive categories with clear coding instructions, they will have to extract raw text fragments and defer the category definition to the coding stage.

In addition, in one-shot reviews, some of the benefits of extracting raw text fragments and separating the categorization from the extraction disappear, and the remaining benefits may not outweigh the costs. Conversely, when conducting a living review, being able to code extracted text fragments again at some point in the future using a different perspective, or having different coders code the text fragments with different goals and instructions can be useful.

In any case, given the importance of this decision, it is worthwhile to carefully document for each entity what the justifications are for its chosen position on the categorization-coding continuum. Later in the process, it is likely you will forget those, and you may even regret your decision for one or more entities – so future you will be grateful for reminders of why that position seemed like a good idea at the time.

7 Planning the Screening

[[Still have to develop this into a section]]

- 1) download all hits of the queries in the various databases (and through the various interfaces) as .bibtex (or .ris) files to your PC;
- 2) import these and deduplicate these (you can do this with a reference manager, or with the metabefor functions, which of course is what I always do because it's more transparent and efficient, and easy to re-run if you slightly change a query)
- 3) write the merged file to disk and send it to all screeners;
- 4) make sure screeners can only see title, keywords, and abstracts, and are blinded from authors, journal, and year etc;
- 5) let screeners indicate for each entry why it is excluded (based on a progressive list of exclusion criteria, that is based on your extraction scripts), or, for entries they cannot exclude, indicate inclusion;
- 6) if you have a lot of hits (thousands), usually you first screen based on title only, and only in the second round, on abstract for those entries that could not be excluded based on title;
- 7) after screening based on abstracts, acquire full-texts and screen those again
- 8) then you have your list of included sources
- 9) something is only excluded if all two/three/... screeners exclude it (the reason can be different; but if one screener fails to exclude, it's retained for the next step)

8 Planning the Search

8.1 Query Crafting

Running your query is the first operational step of your systematic review: it's often one of the first things you do after you publicly froze your preregistration. In that sense it's kind of exciting, but ironically normally the results you obtain will not be surprising, since you repeatedly test your query while crafting it.

8.2 Queries as logical expressions

A query is a logical expression that specifies the conditions that must be met for bibliographic records to be returned by the interfaces that you use to search the bibliographic databases (see below). You first craft this query in a conceptual form, not worrying about the syntax that you will have to use to specify your query in a way the different interfaces can parse.

The simplest queries typically bind together sets of synonyms using the logical conjunction operator (often represented by AND, &, or &&). Each set of synonyms binds together various terms using the logical disjunction operator (often represented by OR, |, or ||). For example, imagine we're doing a systematic review on the determinants of ecstasy use. In that case, a simple query could be:

```
("determinant" OR "determinants" OR "correlate" OR "correlates") AND ("ecstasy" or "XTC" or
```

This query has two terms. We could label the first “determinants”, since it is intended to capture all synonyms for “determinants” (it does so badly, since I wanted to keep this example short; such lists of synonyms are normally much longer), and we could label the second “ecstasy”, since its task is to match all records that contain a word for “ecstasy” (again, doing so badly to enable a brief example).

Using these two logical operators, it's also possible to build more complex queries. For example, if we would know enough about substance use to realise that the determinants of trying out (i.e. “initiation” of ecstasy use) are different from what you'd find if you do a determinant study into the determinants of “using ecstasy”, the second term would become more complex:

```
("determinant" OR "determinants" OR "correlate" OR "correlates") AND (("ecstasy" or "XTC" or
```

The complexity of the query you end up with is often related to the complexity or “subtlety” of your goal or research question. If you’re conducting a scoping review, where you aim to map out the literature on a specific topic, you will generally have simpler queries than when you have a specific narrow research question.

Query complexity is often also related to the richness of the literature. If the literature on a topic is very extensive, the review may become unfeasible if you use a very simple, broad query: you might obtain tens of thousands of hits without the resources to screen all of those. Similarly, if you’re surveying a smaller literature, you can afford to have a less sophisticated query. Since screening costs a lot of time, it usually pays off to spend a lot of time developing your query so that you minimize the number of irrelevant hits.

8.3 Database fields

In addition to the terms themselves, you can specify the fields you want to search. For example, you can search all text fields (the default in most interfaces if you don’t specify one or more fields), or only the title field, or the title and the abstract and the keyword fields, et cetera. Usually you will want to search the titles at the bare minimum; and unless you are confident of relatively standardized vocabulary in a field, you’ll often want to add the abstract and keyword fields. Including fields like journal name, authors, or affiliations usually doesn’t make sense, so omitting explicitly specified field names is very rare.

Sometimes interfaces will allow you to specify multiple fields in a query, for example, indicating that a search term (e.g. a set of synonyms) can occur in either the title or the abstract; but often that’s not possible, and you have to duplicate parts of your query. This can cause queries to grow exponentially, and this is one of the reasons why it is important to craft your query on the conceptual level before starting the translation into the interface languages.

8.4 Wildcard characters

The query languages used by each interface have many advanced features that you can use to build powerful queries, and it is worthwhile diving into those. In addition to logical and other operators, another category of such features is wildcard characters. For example, the asterisk (*) can often be used to signify “zero or more alphabetic characters”, and the question mark (?) can often be used to signify “zero or one alphabetic characters”. This allows you to rewrite this query fragment:

```
"behavior" OR "behaviors" OR "behavioral" OR "behaviour" OR "behaviours" OR "behavioural"
```

into this much shorter fragment:

```
"behavio?r*"
```

Because such operators differ per interface, it usually pays off to obtain a thorough understanding of the capabilities of each interface you will use before starting to craft your query (or while doing so), since you will want to create a query that's as powerful and versatile as you can, but you will have to do this within the constraints of the query languages of the interfaces you'll use.

8.5 Team Consensus and Expert Consultation

It is important to achieve consensus with the team about the query before you finalize your preregistration and then run your query "for real". If you miss important keywords, that can be a very expensive oversight to correct later on (depending on how smartly you designed your screening procedure; see below). For this reason, it is common to involve experts outside the research team to consult on the lists of synonyms and the logical structure of the query.

8.6 Databases versus Interfaces

Once you formulated your conceptual query, you can start translating it into the languages that the interfaces of the database you will use can understand. This language is generally specific to each interface. An interface is the application that performs the searches in the bibliographic databases for you and allows you to export the results in whichever format you want to use.

For example, PsycINFO is a bibliographic database maintained by the American Psychological Association. The APA keeps track of new articles that appear and adds them to PsycInfo. This database is accessible through various interfaces, and different institutions will have licenses with different interface providers. PsycInfo, for example, can be accessed through Ebsco, Ovid, and ProQuest. Ebsco, Ovid and ProQuest use different interface languages, so the syntax and operators you have to use to formulate your query will be different.

Those interfaces are often (but not always) maintained by different organisations than those maintaining the databases. Sometimes, a database maintainer offers its own interface: PubMed is a good example of this. However, usually an interface is developed by a different organisation that then provides access to multiple databases through their interface.

This has a number of benefits. One is that once you're familiar with a given interface, you can use those skills to search multiple databases. For example, your institutions may provide

access to PsycInfo, MedLine, and PsycArticles through an Ebsco interface. It also allows you to search those databases simultaneously.

It also has a number of drawbacks. First, different interfaces work differently. The available operators, the symbols representing those operators, and the syntax you have to use to build a search query therefore differs per interface. If you want to search for a word, say “meta-analysis” in an article title, sometimes you indicate this by saying "meta-analysis" IN TI, and sometimes by saying TI("meta-analysis").

Second, the fields that exist differ per database. If you do search in multiple databases using the same interface, it is very important to clearly keep in mind which fields you search.

As a consequence of this heterogeneity in interface languages, once you crafted your conceptual query, you have to translate it into each interface’s language. Depending on how many fields you want to search and on the features of each language, this can explode your query into quite unwieldy strings of characters. Make sure to document both the conceptual query and the final query you input into each database/interface combination.

Therefore, if you conduct a systematic review, it is important to always preregister both the database(s) you plan to use and the interface(s) you plan to use. In addition, it is important to document the search query you use in every interface/database combination.

8.7 “Smart” searching

When conducting a systematic review, make sure to disable all “smart” searching features of the interface(s) you use. These features expand your query by including other synonyms. However, of course, this “smart” searching algorithm is in fact dumb: it cannot understand your goal(s) and/or research question(s), and so it will simply explode your query to find many more hits, the vast majority of which will be irrelevant to your goals/questions, because after all, you crafted a well-thought-through query.

A second problem of “smart” searching is that it is not replicable, since the algorithms implemented by these interface maintainers are adjusted over time. Since you cannot encode a “smart search version” parameter in your query specification, it’s not possible to solve this. As a consequence, using “smart” searching in effect renders your systematic review unsystematic: it can no longer be reproduced by other researchers — and worse, by yourself in the future.

Since systematic reviews typically take a year and often longer (see <https://predicter.org/>), you will often have to repeat your query towards the end, screening the additional hits, extracting entities from the additional inclusions, and re-running your analysis script. If your query was applied using “smart” searching, the results in this repeated query execution can change unpredictably.

Therefore, never use “smart” searching in the final query you will use (and freeze in your preregistration). You *can* use it while crafting your query, to find additional sources to include,

inspect the titles and abstracts for search terms you may have missed (people use the weirdest synonyms at times), and improve your conceptual query accordingly.

8.8 Query validation

Usually, you'll already have a few sources (e.g. articles, book chapters, etc) that you know you will be including in your systematic review. While testing and perfecting your query, you'll usually use these to check whether your query "works": whether it finds the articles you know it's supposed to find. If it fails to find one or more, then check whether it's supposed to find it: all bibliographic databases have a limited scope, and so the source might simply not exist in that database (easily checked by entering its title as a query). If it was supposed to be included in the hits but wasn't, then your query is missing one or more synonyms, so add those.

A quick way to check whether a given source is included in your query is by combining it with your query: basically create a "single use query" that combines the source's title (or DOI, or ISBN, or any other unique identifier for the source) with your query using the AND operator.

8.9 Exporting query hits

Once you ran your queries, you will need to download the hits: i.e. the identified bibliographic records. There is usually a set of formats available: a very common format that is generally well-supported is RIS (a format developed by "Research Information Systems"), and another good choice is BibTeX. Before deciding on the format, make sure you know how you want to conduct your screening. Ideally, you will be able to easily repeat your query later, either when you revise the manuscript to make sure your findings aren't outdated; if you updated your query because you discovered you made a mistake; or in the case of living reviews, when you want to update the review.

9 (Pre)registrations

It is best practice to (pre)register systematic reviews. This has a number of benefits, one being that others can find out that you're doing a systematic reviews, facilitating collaboration and potentially preventing double work. Conversely, before starting to plan a systematic review, you will typically want to check existing preregistration registries to make sure nobody else started the same review a year ago and is already almost done.

There is an extensive (pre)registration form for systematic reviews, the Inclusive Systematic Review Registration Form, that is available in [this Google Doc](#), in [this MetArXiv preprint](#), and in [the preregr package](#). Using this form can be a useful help while planning your systematic review.

Part II

Execution

10 Executing the Search

10.1 Running your queries

10.2 Exporting query hits

Once you ran your queries, you will need to download the hits: i.e. the identified bibliographic records. There is usually a set of formats available: a very common format that is generally well-supported is RIS (a format developed by “Research Information Systems”), and another good choice is BibTeX. Before deciding on the format, make sure you know how you want to conduct your screening. Ideally, you will be able to easily repeat your query later, either when you revise the manuscript to make sure your findings aren’t outdated; if you updated your query because you discovered you made a mistake; or in the case of living reviews, when you want to update the review.

10.3 Importing your search hits

10.4 Deduplicating your search hits

11 Executing the Screening

This chapter still has to be written.

12 Executing the Extraction

12.1 Data management

12.2 Extracting entities

12.3 Validation of extracted entities

12.4 Contact with authors

13 Executing the Synthesis

This chapter still has to be written.

Part III

References and Appendices

14 Example Projects

14.1 NITRO

The best example project to check out might be the Narrated Illustration of a Transparent Review Outline, or NITRO. NITRO is a bare-bones but narrated illustration of an (excessively simple) scoping review.

It is hosted at <https://gitlab.com/sci-ops/narrated-illustration-of-a-transparent-review-outline>, which is a Git repository. That means that you can easily download all files (for example in [this zip archive](#)) or clone the project. The rendered R Markdown file is served by GitLab Pages at <https://sci-ops.gitlab.io/narrated-illustration-of-a-transparent-review-outline>.

14.1.1 Real-life projects

In addition, this is a list of example `metabefor` projects. They vary in scope: for example, some are bachelor's or master's thesis projects, others are done in the context of a PhD. thesis or are community-run living reviews.

- Drugs and Crime Scoping Review, [GitLab](#)
- Habituation versus Inhibition in Exposure Therapy, [OSF](#)
- A Systematic Review of Works in Quantitative Ethnography, [OSF](#)
- Extending the Earcheck Living Review, [OSF](#), [GitLab](#), [GitLab Pages](#), [Rxs specification](#)

15 Glossary

This glossary contains definitions of terms used in when describing evidence synthesis (e.g., systematic reviews) as well as terms introduced by `metabefor`.

Container entity An *entity* that does not store a value (i.e. cannot itself be extracted) but serves to organise other entities in a hierarchy. For example, a container entity could be “Provenance”, containing “regular” entities “Authors”, “Region”, and “Date”. For the latter three, values are extracted, and though those three together can form a source’s provenance, the entity provenance itself has no value: it just contains the other three entities.

Cluster A set of entities specified in a list in a parent entity: see *clustering entity* and *clustered entity*.

Clustered entity An *entity* that is specified in a list within a *clustering entity*. To specify that an entity is a value list entity, specify as its parent entity a *clustering entity*. Clustering entities are presented in a list in the `Rxs template` (within the parent entity). The coding guides that are normally presented above and below every entity are then concatenated and shows after it. This has two drawbacks. First, if word wrapping is active, it makes the `Rxs` file look quite messy and potentially intimidating. Second, since the descriptions and examples are less salient, this is a bit less straightforward for extractors, so may require more intensive training.

Clustering entity An *entity* that as its value contains a list of *clustered entities*. To specify that an entity is a clustering entity, set the `list` column in the *Rxs specification* spreadsheet to `TRUE`. All entities that specify that clustering entity as parent will then be presented in a list within that entity in the `Rxs template`. They are useful to efficiently extract closely related information. For example, characteristics of a variable that is reported on in a source; various possible effect size measures that could be reported to describe an association; or closely related characteristics of a procedure that is described in a source.

Entity Something that can be extracted from a *source*: examples are effect sizes, sample sizes, study design characteristics, author names, publication dates, journal names, whether sampling strategies were justified, or the literal text used to phrase the main conclusion. Once an *extraction script* has been parsed, entities are also commonly referred to as *fields* that hold *values*. However, note that entities can be repeating: they refer to a type of thing that can be extracted, not to any specific thing that has been extracted, whereas a field always holds a specific extracted value. For example, it is possible to specify a *clustering entity* called `result_association` used to extract various effect sizes or other statistics that may be specified to describe an association between two

variables, each as a *clustered entity*, e.g. entities called `pearson_r` and `cohen_d`. In any given extraction script, multiple instances of that `result_association` entity can be extracted, each being represented by a different `entity node`, and each of those *clustering entities* holding multiple *clustered entities* or *fields*. Each entity node holds a `pearson_r` field and a `cohen_d` field, but in the *extraction script specification*, only one entity is specified for each of `result_association`, `pearson_r`, and `cohen_d`. To refer explicitly to the entity as a “thing that can be extracted”, you can use the terms *entity specification* or *extractable entity*. To refer explicitly to an entity that has been extracted, use `field` or `extracted entity`.

Entity container See *container entity*.

Entity identifier The unique identifier of an *entity* as specified in the `identifier` column of the *extraction script specification*.

Entity node In the `data.tree` object representing each extracted source, *entities* are stored as nodes in a hierarchical tree. In this tree, each entity corresponds to one or more entity nodes, except for *clustered entities*, which are stored as *values* in a list in the entity node representing their *clustering entity* (i.e. their parent entity).

Entity node identifier The unique identifier of an *extracted entity* (i.e. an instance of an *entity specification*)

Entity specification A way to explicitly refer to an entity in its generic definition, as a class of things that can be extracted, as opposed to as a *field* or *entity node*, which represents an extracted entity. This distinction is important because entities can be repeated (see *entity*).

Extractable entity See *entity specification*.

Extracted entity A way to explicitly refer to an entity that has been extracted, as an instance of the *entity specification* or *extractable entity* (a generic description of a class of things that can be extracted). Extracted entities are *entity nodes* unless they are *clustered entities*, in which case they are stored as *fields* holding *values* in a *list* in their parent entity (i.e. their *clustering entity*). Extracted entities that are entity nodes are either *container entities* (if they don’t store a value themselves, but instead are used to organise other entities), *fields* (if they hold a single *value*), or *clustering entities* (if they hold a *list* specifying the values for multiple *clustered entities*).

Extraction The act of registering an *entity* in an *extraction script* (or extraction form, if `metabefor` isn’t used) in the form of a text string or an entity conform the coding instructions (as specified in the *Rxs specification* if `metabefor` is used).

Extraction script (.Rxs file) An R Markdown file with the extension `.Rxs.Rmd` that is a completed *Rxs template*, and as such, set up such that it can be parsed by `metabefor`.

Extraction script specification A spreadsheet specifying which entities to extract, how they are hierarchically structured, which *value templates* they use, and their extraction instructions (or coding instructions).

Field A synonym of entity that is used when it is referred to as a specific value holder in the context of a completed extraction script. For example, when in an *extraction script specification* an entity is specified to extract the source’s publication date with entity identifier `pub_date`, in the parsed *extraction script*, the value of that date can be referred

to as being stored in the `pub_date` field. A field represents an instance of an *entity*: see that definition for more details.

List A list of multiple *clustered entities*. Lists are an efficient way to extract closely related entities, such as measurement details about a given variable, descriptives for an outcome, or various effect size metrics that may be reported.

Rxs file A file with an R extraction script. One example is an *Rxs template* as produced by `[metabefor::rxs_fromSpecifications()]`, but usually these Rxs files will be completed versions of that template, each containing one or more extracted *entities* from a *source*. Rxs files have the extension `.Rxs.Rmd`.

Rxs specification See *extraction script specification*.

Rxs template The structured R Markdown file produced by `metabefor` after it parsed the *Rxs specification* spreadsheet.

Source Something that *entities* can be *extracted* from, such as an article, book, case law, report, webpage, or other source.

Value A value; usually a single number or character string, but values can be more complex, too, e.g. vectors of numbers or strings, or even tables (matrices or arrays).

Value list A list of *values*; a synonym for all *extracted clustered entities* stored in an *extracted clustering entity*.

Value templates In an *Rxs specification*, a template for a data type that can be extracted, specifying the values that are allowed, the default value, instructions, and validation rules.

16 Notes