Rveg User Manual

Version 0.2.0

Přemysl Král and Jan Douda

Contents

1 Introduction **3**

1.1 Install and open the guide . . . 3

1.2 How an Rveg database is stored . . . 3

2 Digitize your first relevé **4**

2.1 Open the console menu and enter the header . . . 4

2.2 Enter species in a layer . . . 5

2.3 Find a taxon code and correct an entry . . . 6

2.4 Add shrubs and revise herb covers . . . 6

2.5 Inspect, add another plot, or close . . . 7

3 Edit and manage data entry **7**

3.1 The `addReleve()` arguments . . . 7

3.2 Default header fields . . . 8

3.3 Layers and cover scales . . . 9

3.4 Main menu reference . . . 9

3.5 Reopen a database and repair a plot . . . 9

3.6 Separate header and species entry . . . 10

3.7 Enter a species across several plots . . . 11

3.8 Start with a project-specific header . . . 11

3.9 Register a taxon locally . . . 11

4 Work with checklists and loaded data **12**

4.1 Choose a checklist . . . 12

4.2 Create and use a custom checklist . . . 13

4.3 Load the data with `RvegLoad()` . . . 14

5 Inspect, consolidate, and merge databases **17**

5.1 Inspect metadata and convert cover with `RvegCheck()` . . . 17

5.2 Combine species or layers with `RvegCombine()` . . . 17

5.3 Append surveys with `RvegMerge()` . . . 18

6 Exchange data with Turboveg and JUICE **19**

6.1 How vegetation layers are translated . . . 19

6.2 Import with `TvToRveg()` . . . 20

6.3 Export with `RvegToTv()` . . . 21

6.4 Export with `RvegToJuice()` . . . 22

7 Classify vegetation plots **22**

7.1 Prepare the database and choose a method . . . 22

7.2 The `RvegClassify()` arguments . . . 23

7.3 Run the bundled methods on the same plots . . . 24

7.4 Read ESy results . . . 25

7.5 Read VegKey results . . . 26

7.6 Diagnose taxon matching . . . 27

8 Write and test custom ESy definitions **28**

8.1 The `CreateESyRules()` arguments . . . 28

8.2 Write conditions . . . 29

8.3 Build a small rule set and inspect matches . . . 30

9 Build and use a VegKey model **32**

9.1 The `CreateVegKey()` arguments . . . 32

9.2 Train from labelled relevés . . . 32

9.3 Train from a synoptic table . . . 34

9.4 Classify with your model and check the result . . . 35

10 Troubleshooting and project resources **35**

11 References **36**

1 Introduction

Rveg records, edits, manages, and classifies phytosociological relevés in R. A relevé combines plot information, such as locality and plot size, with the taxa and their observed cover in each vegetation layer. You can digitize field sheets in the R console, exchange plot data with Turboveg and JUICE, and bring the resulting tables into an R analysis. Rveg 0.2.0 also supports formal Expert System definitions and a VegKey-inspired classifier based on taxon constancies.

Start with your first relevé to follow an entire field sheet through the console. The subsequent chapters cover each exported function, including input arguments, example output, and ways to inspect the result. The classification chapter starts with the bundled methods before showing how to write your own definitions and train a model.

1.1 Install and open the guide

For the CRAN release, use the first command below. To install from a local source tarball, substitute its path in the second command. The version reported by `packageVersion()` identifies the implementation that will run the examples.

```
install.packages("Rveg")

install.packages("Rveg_0.2.0.tar.gz", repos = NULL, type = "source")

library(Rveg)
packageVersion("Rveg")
vignette("Rveg")
help("addReleve", package = "Rveg")
news(package = "Rveg")
```

To install the development version, use `remotes::install_github("sesitcsl2/Rveg", build_vignettes = TRUE)`. Building the PDF vignette from source requires Pandoc and a LaTeX installation. Run console dialogues in an interactive R or RStudio console. Other R examples can run from a script. Examples that create files use a temporary directory unless they explicitly show a project path.

1.2 How an Rveg database is stored

Rveg creates two CSV files for each database. It adds `HEAD.csv` and `REL.csv` to the database name you supply. Together, these files form the Rveg database.

Filename suffix	Rows	Plot columns
`HEAD.csv`	Header fields such as `ID`, `DATE`, and `PlotSize`. The field names are in `ShortName`.	One field value per plot.
`REL.csv`	Taxon and layer in a single key, such as `TRIFPRA_1`. The keys are in `ShortName`.	One cover value per taxon and layer.

In each file the columns `X1`, `X2`, and so on identify the same plots. A header row such as `FieldCODE` holds the survey identifier you entered. Use that field to keep a stable link to your field sheets when a database is combined, filtered, or reindexed. The following abbreviated tables show the orientation on disk.

```
HEAD                                  REL
ShortName  X1          X2             ShortName  X1  X2
ID         1           2              SAMBNIG_2   2   0
DATE       2026/05/22  2026/05/23     ARRHELA_1  35  18
FieldCODE  M01         M02            TRIFPRA_1  30   0
PlotSize   25          25             RUMEACE_1   5   7
```

Here `TRIFPRA_1` is *Trifolium pratense* in the herb layer. `SAMBNIG_2` is *Sambucus nigra* in the shrub layer. A zero represents absence for that taxon and layer. A blank header denotes a value left unrecorded. Total layer cover entered in the header is a field observation and can differ from the sum of species covers.

The lines beginning with `#` form the database metadata. They store the project title and description, the checklist, the number of plots, any project-specific taxa, the database ID, creation and last-change times, and the Rveg version recorded in the files. Keep both files together when moving or copying the database.

Use `addReleve()` to make changes to a working database. Avoid editing the original CSVs in a spreadsheet editor. Saving from a spreadsheet can change delimiters, dates, or text encoding, or damage the metadata, row keys, and matching `X` columns. A damaged pair may fail to load in Rveg.

Once data entry is complete, use `RvegLoad()` to export separate tables for spreadsheet work. Keep the original database for `addReleve()`. The tables exported from `RvegLoad()` are for analysis and cannot be edited through `addReleve()`. See Export tables for a spreadsheet.

When a function reads or creates an Rveg database, pass its name without `HEAD.csv` or `REL.csv`. Include the folder in the path when the database is outside your working directory. For a local file supplied to `tv`, `rules`, or `model`, give the complete filename.

2 Digitize your first relevé

2.1 Open the console menu and enter the header

Follow this example with the default Czech checklist. The call creates a new database in the current working directory.

```r
library(Rveg)

addReleve(
  save = "meadow_survey",
  metadata = c("Meadow survey", "Permanent plots in Prague-Suchdol")
)
```

Rveg creates the CSV pair and opens the `$Rveg:` menu. Enter `Y` to add a complete plot. The console accepts lower-case responses for menu choices. The transcript shows the prompts together with what you type. Use the date and site values from your own field sheet when digitizing your data.

```
$Rveg: Y
DATE? 2026/05/22
SpringDATE?
```

```
LOCALITY? Prague-Suchdol
FieldCODE? M01
Authors? PK
PlotSize? 25
Latitude? 50.12981
Longitude? 14.37081
Accuracy? 5
CRS? WGS84
Slope?
Exposure?
E3? 0
E2? 5
E1? 75
Ejuv? 0
E0? 0
Note?
```

Rveg fills the numeric `ID` automatically. The survey code `M01` is kept in `FieldCODE`. Header values are entered as text. Use a consistent date format, coordinate reference system, and convention for missing values throughout a survey. Leave `SpringDATE` blank when you have no spring visit. Record the estimated total cover of each vegetation layer in its header field before entering the individual taxon covers.

2.2 Enter species in a layer

After the header, choose `Y` to add a layer. Select `1` for herbs and `P` for percentage cover. Enter the short code and estimated cover, then confirm the name displayed by Rveg.

```
AddNewLayer? (Y/N) Y
Select Layer (3,2,1,J,0) 1
P - percentage, BB - Braun-Blanquet scale, CS - custom scale P
AddSpecies? (GenuSpe / SEARCH / INSERT / N) TRIFPRA
Abundance? (%) 30
[1] "Trifolium pratense"
CorrectName? (Y=accept / N=reject) Y

AddSpecies? (GenuSpe / SEARCH / INSERT / N) ARRHELA
Abundance? (%) 25
[1] "Arrhenatherum elatius"
CorrectName? (Y=accept / N=reject) Y

AddSpecies? (GenuSpe / SEARCH / INSERT / N) RUMEACE
Abundance? (%) 5
[1] "Rumex acetosa"
CorrectName? (Y=accept / N=reject) Y
```

The prompt shows the records in the layer you are editing after each accepted entry. Codes typically use four letters from the genus and three from the epithet. Codes can differ when names would otherwise produce the same abbreviation. Confirm the full name each time.

2.3 Find a taxon code and correct an entry

Suppose you enter `BELLPER` for *Bellis perennis*. When Rveg cannot resolve the code, use `SEARCH` at the correction prompt. Type the start of the full name and read the `ShortName` column in the result. You can also run `SEARCH` immediately at the main species prompt.

```
AddSpecies? (GenuSpe / SEARCH / INSERT / N) BELLPER
Abundance? (%) 4
Code BELLPER not found.
SpeciesCode? (GenuSpe / SEARCH / INSERT / N) SEARCH
SpeciesFirst3letters? (eg.Che / F=cancel) Bellis
[Search results include BELSPER, Bellis perennis.]
Enter SpeciesCode (or N to cancel): BELSPER
[1] "Bellis perennis"
CorrectName? (Y=accept / N=reject) Y
```

The correction keeps the cover already entered for this record. Enter `N` at the correction prompt to cancel the record, or `F` at the search-text prompt to leave the lookup. If you reject a valid code at `CorrectName?`, Rveg takes you to the same correction dialogue.

2.4 Add shrubs and revise herb covers

Finish the current layer with `N` at the species prompt. Add a shrub layer (`2`) and choose `BB` for a Braun-Blanquet entry. The example uses `+`, which Rveg stores as 2%.

```
AddSpecies? (GenuSpe / SEARCH / INSERT / N) N
AddNewLayer? (Y/N) Y
Select Layer (3,2,1,J,0) 2
P - percentage, BB - Braun-Blanquet scale, CS - custom scale BB
AddSpecies? (GenuSpe / SEARCH / INSERT / N) SAMBNIG
Abundance? (0,R,+,1,2,M,A,B,3,4,5) +
[1] "Sambucus nigra"
CorrectName? (Y=accept / N=reject) Y
AddSpecies? (GenuSpe / SEARCH / INSERT / N) N
```

To revise *Arrhenatherum elatius*, select the herb layer again. Enter its code with the new cover and confirm. Rveg replaces the previous cover for the same taxon and layer. In the same dialogue, enter `0` for `RUMEACE` to mark *Rumex acetosa* as absent from this plot in the herb layer. A record of the same taxon in another layer has its own key.

```
AddNewLayer? (Y/N) Y
Select Layer (3,2,1,J,0) 1
P - percentage, BB - Braun-Blanquet scale, CS - custom scale P
AddSpecies? (GenuSpe / SEARCH / INSERT / N) ARRHELA
Abundance? (%) 35
[1] "Arrhenatherum elatius"
CorrectName? (Y=accept / N=reject) Y
AddSpecies? (GenuSpe / SEARCH / INSERT / N) RUMEACE
Abundance? (%) 0
[1] "Rumex acetosa"
```

```
CorrectName? (Y=accept / N=reject) Y
AddSpecies? (GenuSpe / SEARCH / INSERT / N) N
AddNewLayer? (Y/N) N
Species richness
2 1
1 3
$Rveg:
```

The richness display gives the counts for populated layers. Here it shows one shrub-layer record and three herb-layer records. The layer counts can count a taxon in more than one layer if it occurs there.

2.5 Inspect, add another plot, or close

At `$Rveg:`, use `PRINTREL` and `PRINTHEAD` to review your first plot. Rveg prints the stored tables. The composition result should include these rows, with display details depending on the console.

```
$Rveg: PRINTREL
  ShortName X1
  SAMBNIG_2  2
  ARRHELA_1 35
  BELSPER_1  4
  TRIFPRA_1 30

$Rveg: PRINTHEAD
  ShortName          X1
  ID                  1
  DATE       2026/05/22
  SpringDATE
  LOCALITY   Prague-Suchdol
  FieldCODE         M01
  ...
```

Use `Y` for another complete plot. When a prompt shows a previous value after `RE <-`, enter `RE` to reuse that value. For example, reuse the locality for two plots sampled at the same site and enter a new `FieldCODE` for the second plot. Press Enter for a blank value. When you are finished, enter `Q` (or `N`) at the main menu. The files can now be reopened with `addReleve()` or loaded with `RvegLoad()`.

```
meadow <- RvegLoad("meadow_survey", variation = 4)
meadow$HeaderDATA["X1", c("DATE", "FieldCODE", "PlotSize")]
meadow$RelDATA["X1", c("Trifolium pratense_1", "Sambucus nigra_2")]
```

3 Edit and manage data entry

3.1 The `addReleve()` arguments

```
addReleve(
  database = "NEW", save = "default", checklist = "default",
  customhead = NULL, extrahead = NULL, metadata = NULL
)
```

Argument	How it is used
`database`	`"NEW"` creates a database. To reopen one, supply its name or path.
`save`	Name or path for the saved database. Use the same value as `database` to edit in place, or a new name to work on a copy. `"default"` saves to the R session's temporary directory.
`checklist`	Built-in name or local text-file path. New projects default to `cz_dh2012`. On reopening, Rveg prefers the recorded checklist when it can resolve it.
`customhead`	Defines the complete set of user header fields for a new database. Rveg supplies `ID` first.
`extrahead`	Appends fields to the default or existing schema. For a new database with `customhead` supplied, include all the desired fields in `customhead`.
`metadata`	A two-element character vector for the title and description of a new project. Use `RvegCheck()` to fill missing values in an existing project.

The main console menu stays active until you leave it. Rveg writes the header before the composition when you add a complete plot and writes the completed plot when you leave its layer dialogue. If a session is interrupted, compare the two tables when reopening. Close other programs that have the output files open, especially before saving to an existing database.

3.2 Default header fields

Field	What to record
`ID`	Internal numeric identifier generated on entry. Preserve it when editing by hand.
`DATE`, `SpringDATE`	Main and spring survey dates, for example `2026/05/22`.
`LOCALITY`	Site or locality description.
`FieldCODE`	Project identifier linked to the original field sheet.
`Authors`	Recorder name or initials.
`PlotSize`	Sampled area, conventionally in square metres.
`Latitude`, `Longitude`	Plot location in the stated coordinate system.
`Accuracy`, `CRS`	Coordinate accuracy and coordinate reference system.
`Slope`, `Exposure`	Slope in degrees and aspect using a consistent convention.
`E3`, `E2`, `E1`	Total tree, shrub, and herb cover.
`Ejuv`, `E0`	Total juvenile and moss cover.
`Note`	Additional field notes.

Imported datasets may have different header fields. Use `PRINTHEAD` in the console or inspect the names of the plot-by-field `HeaderDATA` table before writing analysis code or ESy rules.

3.3 Layers and cover scales

Rveg code	Layer	Typical use
`3`	Tree	Trees recorded in the canopy.
`2`	Shrub	Shrubs or woody taxa recorded in the shrub layer.
`1`	Herb	Herb-layer taxa.
`J`	Juvenile	Juveniles recorded separately.
`0`	Moss	Moss-layer records.

With `P`, enter a number from 0 to 100. With `BB`, Rveg applies the following built-in conversion. The import function also recognizes `2M`, `2A`, and `2B` as aliases of `M`, `A`, and `B`. The interactive entry prompt lists the shorter forms.

Score	0	R	+	1	2M / M	2A / A	2	2B / B	3	4	5
Stored %	0	1	2	3	4	8	15	18	38	63	88

With `CS`, values are stored as entered. Keep the field scale and its percentage equivalents in your project notes. The conversion prompt replaces each distinct stored value throughout the dataset. If the same symbol means different things in different surveys, convert the surveys separately before merging them.

3.4 Main menu reference

Command	Effect
`Y`	Add a complete plot, header first and species second.
`ADDREL`	Add only a species column.
`ADDHEAD`	Add only a header column.
`YSP`	Enter species-wise cover for a batch of new plots.
`RREL`	Revise the composition of a selected plot via the layer dialogue.
`RHEAD`	Revise one header field or refill the header of a selected plot.
`PRINTREL`	Print the species and cover table.
`PRINTHEAD`	Print the header table.
`PRINTMETA`	Print metadata, including extra taxa.
`REMOVEREL`	Remove the selected plot from both tables and reindex the `X` columns.
`INFO`	Show current project and session information.
`?`, `H`, `HELP`	Display help in the console.
`Q`, `N`	Save and leave the main menu.

3.5 Reopen a database and repair a plot

The following example creates an edited copy and adds two header fields. Existing plots receive empty entries in the new fields. To edit the original, give `save` and `database` the same value.

```
addReleve(
  database = "meadow_survey",
  save = "meadow_edited",
  extrahead = c("Management", "ObserverNotes")
)
```

To fill the new `Management` field on the first plot, type `RHEAD`, select plot number `1` (column `X1`), confirm the displayed header, and choose single-field repair. Full field names work directly. Use `R` to refill all editable fields or `C` to cancel. In refill mode, `RE` retains the displayed value.

```
$Rveg: RHEAD
ReleveNumber? 1
[Header for X1 is displayed.]
CorrectColumn? (Y/N) Y
Repair mode: (S)ingle field / (R)efill all / (C)ancel ? S
HeaderCharacteristic to repair? Management
Management? Mown annually
[Updated header is displayed.]
save changes? (Y/N) Y
Header updated.
```

To change a species cover on that plot, use `RREL`, select `1`, confirm the header, and enter the corrected value in its layer. For example, the following changes *Trifolium pratense* from 30% to 20% in the herb layer.

```
$Rveg: RREL
ReleveNumber? 1
CorrectNumber? (Y/N) Y
AddNewLayer? (Y/N) Y
Select Layer (3,2,1,J,0) 1
P - percentage, BB - Braun-Blanquet scale, CS - custom scale P
AddSpecies? (GenuSpe / SEARCH / INSERT / N) TRIFPRA
Abundance? (%) 20
[1] "Trifolium pratense"
CorrectName? (Y=accept / N=reject) Y
AddSpecies? (GenuSpe / SEARCH / INSERT / N) N
AddNewLayer? (Y/N) N
$Rveg: PRINTREL
```

Use `REMOVEREL` if an entire plot was entered by mistake. Rveg displays the header for the selected plot before asking for confirmation. After removal, review both tables and any analysis files that use the old `X` keys. For example, if you remove `X2` from a three-plot database, the previous `X3` becomes `X2`. The recorded field codes let you identify the surviving plots.

3.6 Separate header and species entry

`ADDHEAD` and `ADDREL` each add one column to their respective table. If several people collect or transcribe the two kinds of data, enter the columns in the same plot order. The first species column belongs to the first header column, regardless of the order in which you enter each table. Check the counts and print both tables before returning to complete-plot entry. The `Y` and `YSP` paths require the current counts to agree.

For example, to enter headers during a field visit and composition later, use `ADDHEAD` once per field sheet and record its `FieldCODE`. Resume with `ADDREL` in the same order. If a session stops after writing a header but before saving the species column, use `ADDREL` to complete the missing composition in the corresponding position.

3.7 Enter a species across several plots

Use `YSP` when the source is a species-by-plot matrix and each row gives a species, its layer, and the covers in successive columns. Decide on the number and order of the new plots before you start. The scale choice applies to the batch. Enter zero where a species is absent from a plot.

```
$Rveg: YSP
How many releves? 3
P - percentage, BB - Braun-Blanquet scale, CS - custom scale P
Add new species (GenuSpe)/N? TRIFPRA
Select Layer (3,2,1,J,0) 1
[1] "Trifolium pratense"
CorrectName? (Y=accept / N=reject) Y
Relevé 1
Abundance? (%) 30
Relevé 2
Abundance? (%) 0
Relevé 3
Abundance? (%) 12
Add new species (GenuSpe)/N? N
Add headers? (Y/N) Y
```

Repeat the species entry for all rows before answering `N`. Answer `Y` to supply the three headers in the same order. If you answer `N`, use `ADDHEAD` later to enter the matching headers. Use `PRINTHEAD` and `PRINTREL` to verify the result.

3.8 Start with a project-specific header

Use `extrahead` to keep the standard schema and add a field. Use `customhead` when your survey uses a different set of fields. The following project has a small header for monitoring plots and includes a unit code that can later be used to train a VegKey model.

```
addReleve(
  save = "wetland_reference",
  customhead = c("DATE", "FieldCODE", "PlotSize", "VegetationUnit"),
  metadata = c("Wetland reference plots", "Plots with reviewed labels")
)
```

Rveg provides `ID` and then prompts for the supplied fields. To extend this database later, reopen it with `extrahead`. Choose consistent field names before merging databases. For example, `PlotSize` and `area` are separate header fields on merge.

3.9 Register a taxon locally

If a taxon is missing from your checklist, enter `INSERT` in the single-plot species dialogue. Provide the name and a unique seven-character code. Use a code that is distinct from all taxa already recorded in the checklist and in the project's extra taxa. Confirm both the name and code. Then enter the new code with its cover in the usual way.

```
AddSpecies? (GenuSpe / SEARCH / INSERT / N) INSERT
Full species name? (N to cancel): Carex sp. meadow morphotype
Carex sp. meadow morphotype: correct? (Y/N) Y
ShortCode? (GenuSpe / N to cancel): CAREXMO
Carex sp. meadow morphotype; CAREXMO? (Y/N) Y
Added CAREXMO to checklist.
AddSpecies? (GenuSpe / SEARCH / INSERT / N) CAREXMO
Abundance? (%) 3
[1] "Carex sp. meadow morphotype"
CorrectName? (Y=accept / N=reject) Y
```

The extra name and code are saved in the database metadata so that the next session can resolve the taxon. Check them with `PRINTMETA` or `RvegCheck()`. A survey-wide naming list can instead be made with `CreateChecklist()` and reused for multiple new databases.

4 Work with checklists and loaded data

4.1 Choose a checklist

Rveg uses a checklist to translate the seven-character codes stored in the composition table into botanical names. Each code identifies one taxon. The vegetation layer is added separately in the database key. The built-in checklists can be passed by name.

Name	Scope
`cz_dh2012`	Czech vascular plant checklist following Danihelka et al. (2012). Default for a new project.
`cz_kaplan2019`	Czech checklist following Kaplan et al. (2019).
`Czechia_slovakia_2015`	Czechia and Slovakia checklist for matching Turboveg imports.
`wcvp_por`	WCVP-based list for Portugal.
`wcvp_que`	WCVP-based list for Quebec.

Existing databases normally use the checklist recorded in their metadata. If you move a database that uses a local checklist, move the checklist with it and keep a compatible path. You can supply a path through `checklist` if the recorded checklist can no longer be resolved. For a new database or a Turboveg import, `checklist` selects the initial dictionary. Make a copy before adapting a checklist. Changing an established short code breaks the link to composition records that still use it.

To inspect the built-in default without opening a digitizing session, read the tab-delimited dictionary and find the rows you need.

```
default_checklist <- system.file(
  "extdata", "RvChecklist", "cz_dh2012.txt",
  package = "Rveg", mustWork = TRUE
)
dictionary <- read.delim(default_checklist, encoding = "UTF-8")
dictionary[dictionary$ShortName %in% c("BELSPER", "TRIFPRA"),
           c("ShortName", "FullName")]
```

```
#>       ShortName           FullName
#> 1082    BELSPER    Bellis perennis
#> 10426   TRIFPRA Trifolium pratense
```

4.2 Create and use a custom checklist

```
CreateChecklist(specieslist, export = "export")
```

`specieslist` accepts a character vector of botanical names or the name of a UTF-8 text file. Use the text file option for a list with only one name. Give `export` the name you want for the checklist, including a folder if needed. Rveg adds `.txt` and writes a tab-delimited file with `ShortName` and `FullName` columns. The function also returns the checklist table invisibly.

```
species_names <- c(
  "Galium aparine", "Galium x pomeranicum",
  "Primula veris", "Primula veris subsp. veris",
  "Carex flava agg.", "Taraxacum sect. Taraxacum"
)
made <- CreateChecklist(species_names, export = file.path(work_dir, "survey_taxa"))
made[made$FullName %in% species_names, ]
#>   ShortName                   FullName
#> 5   CARE#FL           Carex flava agg.
#> 1   GALIAPA             Galium aparine
#> 2   GALI*PO       Galium x pomeranicum
#> 3   PRIMVER              Primula veris
#> 4   PRIMV-V Primula veris subsp. veris
#> 6   TARASET  Taraxacum sect. Taraxacum
```

Ordinary binomials generally yield four genus characters followed by three epithet characters. Hybrids use a marker such as `*`, infraspecific names can use a dash, and aggregates can use `#`. The generator also supplies genus-level entries such as `Genus species` when applicable. It trims input, resolves normalized duplicates, and adjusts codes when an abbreviation would collide. Use the generated file to look up the actual code, including for sections, hybrids, and names resembling one another.

If your names are already in `botanical_names.txt`, put one full name per line without a column heading. The following call reads that file from your working directory and writes `survey_taxa.txt`. Leave off `.txt` in both arguments to `CreateChecklist()`.

```
CreateChecklist("botanical_names", export = "survey_taxa")
```

To use the checklist, give `addReleve()` its full filename, including `.txt` and the folder if needed. This call uses the checklist created in the working directory.

```
addReleve(save = "my_custom_survey", checklist = "survey_taxa.txt")
```

When preparing a checklist manually, keep the columns tab-delimited, use unique upper-case seven-character codes, and preserve UTF-8 taxon names. Consult the generated file before digitizing if you need to reconcile locally used names. Keep a named copy of the checklist used for each field season.

4.2.1 How the bundled WCVP checklists were made

The Portugal (`wcvp_por`) and Quebec (`wcvp_que`) checklists ship with Rveg and are ready to use. To prepare them, we used the `rWCVP` package to retrieve names from the World Checklist of Vascular Plants by area code. We took the unique names in `taxon_name` and passed them to `CreateChecklist()`. This recipe is for readers who already work with `rWCVP` and its WCVP data. See the rWCVP checklist documentation.

```
portugal <- rWCVP::wcvp_checklist(area_codes = "POR")
CreateChecklist(unique(portugal$taxon_name), export = "wcvp_por")

quebec <- rWCVP::wcvp_checklist(area_codes = "QUE")
CreateChecklist(unique(quebec$taxon_name), export = "wcvp_que")
```

4.3 Load the data with `RvegLoad()`

```
RvegLoad(
  database = "default", checklist = "default",
  customscale = FALSE, variation = 1
)
```

Argument	How it is used
`database`	Name or path of the Rveg database. `"default"` loads the bundled Liberec example.
`checklist`	Checklist name or file path. Rveg prefers a resolvable checklist from the database metadata.
`customscale`	Ask for percentage equivalents of the stored cover values during loading.
`variation`	Choose the structure returned in R.

Variation	Output	When to use it
`1`	One data frame with header rows above composition rows and extra `fullName` and `layer` columns.	Inspect a whole database together.
`2`	List with `HeaderDATA`, raw `RelDATA`, and `meta`. Both tables have plots in columns.	Inspect file structure, unsplit taxon-layer keys, or project metadata.
`3`	Same list, with split `ShortName`, `fullName` and `layer` columns in `RelDATA`. Plots remain in columns.	Work with taxonomy and layers in the original table orientation.
`4`	List with plot-by-field `HeaderDATA`, plot-by-taxon-layer `RelDATA`, and `meta`.	Prepare plot-level analysis or join classification results to plot attributes.

The following examples use the same installed database to show the changes in orientation. The first call prints selected rows from the combined view. In variation 2, the suffix is part of the stored `ShortName`. In variation 3 it has its own column.

```r
combined <- RvegLoad(example_db)
combined[1:3, c("ShortName", "fullName", "layer", "X1")]
#>    ShortName fullName layer   X1
#> 1         ID                   1
#> 2       DATE                2022
#> 3 SpringDATE

raw <- RvegLoad(example_db, variation = 2)
names(raw)
#> [1] "HeaderDATA" "RelDATA"    "meta"
head(raw$RelDATA[, c("ShortName", "X1")], 4)
#>   ShortName X1
#> 1 ATHYFIL_1  0
#> 2 CALLPAL_1  0
#> 3 CARDDEN_1  1
#> 4 CAREELA_1 45

named <- RvegLoad(example_db, variation = 3)
head(named$RelDATA[, c("ShortName", "fullName", "layer", "X1")], 4)
#>   ShortName              fullName layer X1
#> 1   ATHYFIL Athyrium filix-femina     1  0
#> 2   CALLPAL       Calla palustris     1  0
#> 3   CARDDEN     Cardamine dentata     1  1
#> 4   CAREELA           Carex elata     1 45
```

The variants that return a list also expose metadata. For example, use `raw$meta$checklist` to see the recorded checklist and `raw$meta$extra_spec` to review taxa defined within the project.

4.3.1 Use plots as rows

Variation 4 keeps the original `X` keys as row names in both tables. The cover table uses names such as `Carex elata_1`. The suffix keeps different layers in separate columns. For unresolved taxa, column names use the short code in place of the botanical name. Where two generated column names coincide, Rveg disambiguates them with a suffix. Access a column with brackets when its name contains spaces.

```r
plots <- RvegLoad(example_db, variation = 4)
plots$HeaderDATA[, c("LOCALITY", "PlotSize")]
#>    LOCALITY PlotSize
#> X1       CB      200
#> X2       CB      200
#> X3       CB      200
plots$RelDATA[, c("Carex elata_1", "Phragmites australis_1")]
#>    Carex elata_1 Phragmites australis_1
#> X1            45                    10
#> X2            65                    25
#> X3            10                     0

identical(rownames(plots$HeaderDATA), rownames(plots$RelDATA))
#> [1] TRUE
```

```r
plots$RelDATA["X1", "Carex elata_1"]
#> [1] 45
```

Header columns are character values. Convert fields explicitly for analysis. For a plot-level export with column names preserved, add the row name as a field before writing.

```r
site <- plots$HeaderDATA
site$plot_id <- rownames(site)
site$plot_area_m2 <- as.numeric(site$PlotSize)
site[, c("plot_id", "LOCALITY", "plot_area_m2")]
#>    plot_id LOCALITY plot_area_m2
#> X1      X1       CB          200
#> X2      X2       CB          200
#> X3      X3       CB          200
```

When every nonblank cover is numeric, variation 4 returns numeric cover. If there is a symbol such as `+`, the table preserves the original cover values as character strings. A purely numeric ordinal scale can also load as numeric. Confirm the meaning of the values against the scale used on the field sheets before treating them as percentages. Use the original `X` keys to align the two tables before subsetting or joining.

4.3.2 Export tables for a spreadsheet

Once data entry is complete, write the tables from `RvegLoad()` to separate files for spreadsheet work. This example uses `plots` from above and writes the plot keys as a column in each file. Open `plot_headers.csv` and `plot_cover.csv` in your spreadsheet editor.

```r
cover <- plots$RelDATA
cover$plot_id <- rownames(cover)
write.csv(site, file.path(work_dir, "plot_headers.csv"), row.names = FALSE)
write.csv(cover, file.path(work_dir, "plot_cover.csv"), row.names = FALSE)
```

4.3.3 Convert a custom cover scale

The following example assumes the field codes `0`, `1`, `2`, and `3` represent absence, 5%, 20%, and 50%. Run the call interactively and provide one mapping for every distinct stored value. The prompt order follows the data.

```r
converted <- RvegLoad(
  database = "meadow_custom_scale", customscale = TRUE, variation = 4
)
```

```
Percentage value for (0) 0
Percentage value for (1) 5
Percentage value for (2) 20
Percentage value for (3) 50
```

The returned tables contain the converted values. To persist the conversion as a new Rveg database, use `RvegCheck()` with an explicit output name. Keep the original field codes and the conversion table with your project.

5 Inspect, consolidate, and merge databases

5.1 Inspect metadata and convert cover with `RvegCheck()`

```
RvegCheck(database, export = "export", checklist = "default")
```

Use `database` to name the database to check and `export` to name the resulting database. The default destination is temporary. `checklist` supplies a dictionary when the recorded one cannot be resolved. The console reports the project name and description, database ID, recorded Rveg version, dates, and local extra taxa. It offers to fill in missing project fields and asks whether to convert cover.

```
RvegCheck(
  database = "meadow_custom_scale",
  export = "meadow_percentage"
)
```

For a database created in Rveg with a blank project title and description and the example scale above, the dialogue can look as follows.

```
Do you want transfer cover to percentage values? (Y/N) Y
Percentage value for (0) 0
Percentage value for (1) 5
Percentage value for (2) 20
Percentage value for (3) 50
No project name found. Enter? (Y/N) Y
Project name: Meadow survey
No project info found. Enter? (Y/N) Y
Project info: Permanent plots surveyed in May
```

Answer `N` to the conversion question if cover is already in percentages. Reload the output to inspect the edits and confirm the two files can be read together. To combine two taxon rows or move a layer, use `RvegCombine()`.

5.2 Combine species or layers with `RvegCombine()`

```
RvegCombine(database, export = "export", checklist = "default")
```

`database` names the source database, `export` names the saved database, and `checklist` gives the fallback dictionary. This interactive operation applies to all plots within one database. Work with percentage covers. `PRINTREL` displays the composition before or after an operation, and `N` leaves the menu. A successful `LAYER` or `SPEC` operation writes the output pair.

```
RvegCombine(
  database = "woodland_survey",
  export = "woodland_layers_combined"
)
```

For example, suppose *Alnus glutinosa* occurs in both the tree and shrub layers of `X2`, with covers 60% and 23%. Use `LAYER` to move all shrub-layer records into the tree layer.

```
Combine? (LAYER/SPEC/PRINTREL/N) PRINTREL
  ShortName X1 X2
  ALNUGLU_2  0 23
  ALNUGLU_3  0 60

Combine? (LAYER/SPEC/PRINTREL/N) LAYER
Which layer? (3/2/1/0/J) 2
To which layer? (3/2/1/0/J) 3
Combine? (LAYER/SPEC/PRINTREL/N) PRINTREL
  ShortName X1 X2
  ALNUGLU_3  0 69.2
```

Where both records occur in a plot, the probabilistic cover sum estimates the combined cover assuming independent overlap. Rveg rounds the result to two decimal places. If only one record is present, its cover is carried forward as recorded.

$$C_{\text{combined}} = C_1 + C_2(1 - C_1/100).$$

For the alder example, $60 + 23 \times 0.40$ gives 69.2%. The operation changes the species records. Review the header layer totals separately if those totals also need revision.

Use `SPEC` when a taxonomic review calls for one source code to be folded into another existing row. Supply both full taxon-layer keys, including the underscore and layer. The second key is retained. For example, after deciding that an unidentified `Carex` morphotype and *Carex elata* refer to the same taxon, and with both rows present in the database, enter the following.

```
Combine? (LAYER/SPEC/PRINTREL/N) SPEC
Which species? (GenuSpe_L): CAREXMO_1
With which species? (GenuSpe_L): CAREELA_1
Combine? (LAYER/SPEC/PRINTREL/N) PRINTREL
Combine? (LAYER/SPEC/PRINTREL/N) N
```

Check the resulting row and its cover values against the taxa you intended to combine. If a taxon occurs in several layers, decide which source and target keys to combine in each layer.

5.3 Append surveys with `RvegMerge()`

```
RvegMerge(database_1, database_2, export = "export")
```

`database_1` and `database_2` name the databases to merge. `export` names the merged database. The first database supplies the checklist and project metadata for the merged result. The function retains the union of header fields, fills unavailable headers with blanks, and fills unavailable taxon-layer occurrences with zeros. It appends the second set of plots and reindexes the `X` columns. Keep a source or field-code header if you need to trace each output plot back to its original survey. The original header values travel with the plots, so `ID` values can repeat across the merged surveys.

The example below merges a bundled database with itself to make the order and plot-count change visible. A survey merge normally uses different input databases.

```
merged_db <- file.path(work_dir, "merged")
RvegMerge(database_1 = example_db, database_2 = example_db,
          export = merged_db)
merged <- RvegLoad(merged_db, variation = 4)
nrow(plots$HeaderDATA)
#> [1] 3
nrow(merged$HeaderDATA)
#> [1] 6
merged$HeaderDATA[, c("ID", "LOCALITY")]
#>    ID LOCALITY
#> X1  1       CB
#> X2  2       CB
#> X3  3       CB
#> X4  1       CB
#> X5  2       CB
#> X6  3       CB
```

5.3.1 Reconcile checklists before interpreting the merge

When the input checklists differ, Rveg warns and reports the taxon changes from the second database. It compares names and can match normalized spellings, such as certain differences in infraspecific abbreviations. A shared code is reused when it represents the same normalized name. The layer suffix is retained. Where a name has no unique target code or a source code would collide, Rveg retains a separate taxon entry in `extra_spec`, assigning a new code if needed. This preserves observations that would otherwise be folded into a different taxon.

Read the reported mapping and check the extra taxa in the returned metadata. An unresolved source code needs a name in the source checklist or source metadata before it can be reconciled. Duplicate taxon-layer keys can also stop a merge. Confirm that row keys and checklists are valid, cover values use the same scale, and fields with the same name have the same meaning and units. Then compare plot counts and inspect a few expected species and header values in the merged tables.

For example, the Danihelka-based checklist uses `BELSPER` for *Bellis perennis* and the Kaplan-based checklist uses `BELIPER`. If the first database uses `cz_dh2012` and the second uses `cz_kaplan2019`, their herb-layer records for that name are merged into `BELSPER_1` in the output. After creating one plot with each checklist, inspect the corresponding row in the raw merge and verify its two cover values.

```
RvegMerge("survey_dh2012", "survey_kaplan2019", export = "joined_surveys")
joined <- RvegLoad("joined_surveys", variation = 2)
subset(joined$RelDATA, ShortName == "BELSPER_1")
joined$meta$checklist
```

6 Exchange data with Turboveg and JUICE

6.1 How vegetation layers are translated

Rveg	Layer	Turboveg export	JUICE export and classification
3	Tree	t1	2
2	Shrub	s1	4
1	Herb	hl	6
J	Juvenile	jl	7
0	Moss	ml	9

Turboveg import with `Rveglayers = TRUE` maps layer codes starting with `t`, `s`, `h`, `j`, and `m` to the corresponding Rveg layers. For numeric input layers, codes `1`–`3` map to trees, `4` and `5` to shrubs, `6` to herbs, `7` and `8` to juveniles, and `9` to moss. Numeric `0` also maps to moss on Turboveg import. Several incoming sublayers can therefore become one Rveg layer. Check the output before comparing data across workflows that use different layer conventions.

6.2 Import with `TvToRveg()`

```
TvToRveg(tv, export = "export", checklist = "default", Rveglayers = TRUE)
```

Argument	How it is used
`tv`	Path to the Turboveg `.xml` or `.csv` export.
`export`	Name or path of the new Rveg database.
`checklist`	Checklist for linking the imported full names to Rveg codes.
`Rveglayers`	Map to Rveg layers by default. Set to `FALSE` to retain the imported layer labels in the generated keys. Standard Rveg editing and classification expect the standard Rveg suffixes.

In Turboveg, select the plots to export. For XML, choose **Standard XML file** and include the species lookup table. For a spreadsheet, choose **Spreadsheet table**, use **comma-delimited** format, and include the plot headers and species data. The CSV reader expects the combined layout illustrated by the bundled file, including the separator between the header and species sections. Leave ecological indicator tables out of that export. If the CSV structure or character encoding is uncertain, use the standard XML export.

```
examples <- system.file(
  "extdata", "ExampleDB", package = "Rveg", mustWork = TRUE
)

TvToRveg(
  tv = file.path(examples, "tvexport.xml"),
  export = "liberec_from_xml",
  checklist = "Czechia_slovakia_2015"
)
# For the bundled XML, enter BB when asked about the cover scale.

TvToRveg(
  tv = file.path(examples, "tvexport.csv"),
  export = "liberec_from_csv",
  checklist = "Czechia_slovakia_2015"
```

```
)
# For the bundled spreadsheet, enter P.
```

For a name without a checklist match, search by the start of the botanical name and review the suggested mappings. Enter the short code of the correct taxon and affirm the mapping. To preserve an unmatched name as a local taxon, choose `I` and enter a distinct seven-character code. The following illustrates a search for an unmatched *Salix* name. Confirm a correspondence only when the two names denote the same taxon in the source data and chosen checklist.

```
UNKNOWN SPECIES: 'Salix fragilis'
Search with 3 letters + or (I)nsert new code: Sal
[Matching Salix names and codes are displayed.]
Enter species shortcode to select: SALXEUX
Map 'Salix fragilis' -> 'Salix euxina'? (Y/N): Y
```

At the cover prompt, choose `P` for stored percentages or `BB` (`B` also works) for Braun-Blanquet values. When mapped rows share a taxon and layer, their covers are combined. Load the new database and verify the plot count, a few header fields, and the cover and layer of known taxa. Check project-specific names with `RvegCheck()` or `PRINTMETA`.

6.3 Export with `RvegToTv()`

```
RvegToTv(database, export = "export", checklist = "default", ver = 3)
```

Supply the name or path of the existing database and a name for the export. The checklist supplies the botanical names, and Rveg translates the layer codes to Turboveg abbreviations. Use `ver = 3` for one combined `.csv` file and `ver = 2` for the pair of files ending in `R.csv` and `H.csv`. If the export path includes a folder, that folder must already exist. Rveg warns about unresolved species codes.

```
tv3_name <- file.path(work_dir, "turboveg3")
tv2_name <- file.path(work_dir, "turboveg2")
RvegToTv(example_db, export = tv3_name)
RvegToTv(example_db, export = tv2_name, ver = 2)
basename(c(paste0(tv3_name, ".csv"), paste0(tv2_name, c("R.csv", "H.csv"))))
#> [1] "turboveg3.csv"  "turboveg2R.csv" "turboveg2H.csv"
```

6.3.1 Import the two-table export in Turboveg 2

1. Open or create the receiving database. From **Import**, choose **Free format species data table** and select the file ending in `R.csv`.
2. Assign the first column to species names and the second to layer. Check the matched names, then select percentage as the cover scale if the source values are percentages.
3. Select the intended relevés in the preview and finish the species import.
4. Choose **Import** and **Free format header table**. Select the file ending in `H.csv` and assign the first row as field names.
5. Select a plot identifier as the key. For example, map `ID` to `releve_nr` if the IDs match the imported species table. Map remaining fields or allow the creation of new ones as appropriate for that database.

6. Compare imported plot numbers, header values, and several taxon covers with the Rveg source.

In Turboveg 3, use the combined file, which has the header section above the species section. Check the plot labels, layer mapping, and cover units in the importer preview.

6.4 Export with `RvegToJuice()`

```
RvegToJuice(database, export = "export", checklist = "default")
```

The function writes a comma-delimited species spreadsheet (`R.csv`) and a tab-delimited header table (`H.csv`). It uses the recorded checklist when available and inserts the full botanical names. Species cover zeros become dots, and vegetation layers use JUICE's numeric codes. Files use ISO-8859-15. Rveg writes both files and warns if it omits characters outside this encoding. For example, `Příbram` becomes `Píbram` in the export because the encoding has no `ř`. Use `Pribram` in an export copy of your database to keep the complete place name in an unaccented form.

```
juice_name <- file.path(work_dir, "juice_export")
RvegToJuice(example_db, export = juice_name)
basename(paste0(juice_name, c("R.csv", "H.csv")))
#> [1] "juice_exportR.csv" "juice_exportH.csv"
```

6.4.1 Import the composition and header tables in JUICE

1. In JUICE, use **File → Import → Table → from Spreadsheet File** and select the file ending in `R.csv`.
2. Choose a comma as the separator and use the second column for layer information. Inspect the table size, species names, and plotted cover values in the preview.
3. Choose percentage cover if the input database stores percentages. Finish the species import.
4. In the header data import, select the file ending in `H.csv` and use a tab delimiter. Inspect the plot numbering and check that header values attach to the correct species columns.

Both files are numbered consistently for the export. Check the imported associations against a few original field codes. Use different export names if you are producing both Turboveg and JUICE files for the same database.

7 Classify vegetation plots

7.1 Prepare the database and choose a method

Start from an Rveg database with plot keys, a usable checklist, recognized vegetation layers, and percentage cover. A positive cover is an occurrence. Zero means absence. If the source uses Braun-Blanquet or a local ordinal scale, convert it first and retain a record of the conversion. For a quick review, load the plot-by-taxon-layer table with `variation = 4` and check the names, layers, and cover of a few familiar species.

ESy applies explicit conditions to relevés. Its output can show a unique formal unit, several equally prioritized units, no formal match, or an uncertain classification due to missing plot

headers. The package bundles CzechVeg-ESy basic and full rule files. Where a rule file has usable similarity profiles, FPFI adds a ranking based on those profiles.

VegKey ranks units from a reference model using the constancy of the taxa observed in the plot. The bundled model was trained on CzechVeg-OpenStrat. The ranking is useful when the plot shares taxa with several units, and the training counts and taxon diagnostics help judge the support. Interpret the two methods against the scope and nomenclature of the rule file or reference model. Their scores and decisions answer different questions.

If you have	Use
Formal syntaxon definitions	`method = "esy"` with a bundled or custom rule file.
Reviewed labels on reference relevés	`CreateVegKey(database = ...)`, then `method = "vegkey"`.
A synoptic table with unit-by-taxon constancies	`CreateVegKey(synoptic = ...)`, then `method = "vegkey"`.
A reason to review both lines of evidence	`method = c("esy", "vegkey")`, then compare plot-level results.

7.2 The `RvegClassify()` arguments

```
RvegClassify(
  database, method = "esy",
  rules = "CzechVeg-ESy-basic", model = "CzechVeg-VegKey",
  synonym_file = "WCVP-Czech", plot_id = NULL,
  checklist = "default", customscale = FALSE,
  exclude_layers = c("7", "8"), top_n = 3L, min_fpfi = 0,
  candidate_units = NULL, use_dominance = TRUE,
  output = NULL, esy_mode = "separate"
)
```

Argument	How it is used
`database`	Name or path of the database, or `"default"` for the bundled Liberec example.
`method`	`"esy"`, `"vegkey"`, or both as a character vector.
`rules`	One or more bundled ESy names or file paths. Built-in names are `"CzechVeg-ESy-basic"` and `"CzechVeg-ESy-full"`.
`model`	Bundled `"CzechVeg-VegKey"`, a `.rds` path, or the model object read with `readRDS()`.
`synonym_file`	Bundled Czech WCVP synonyms, the optional global table, a local TSV path, or `NULL` to disable synonym lookup.
`plot_id`	Vector of `X` keys, or `NULL` for all plots. Use the keys returned by `RvegLoad(variation = 4)`.
`checklist`	Checklist passed to `RvegLoad()`. The recorded checklist takes precedence when available.

Argument	How it is used
`customscale`	If `TRUE`, ask for percentage equivalents during loading. Use an interactive session.
`exclude_layers`	JUICE codes to omit from ESy and FPFI. VegKey uses its model's stored exclusions.
`top_n`	Maximum number of FPFI and VegKey ranks retained per plot and run. Use a positive integer to see the ranking.
`min_fpfi`	Retain FPFI scores at or above this threshold, among the positive scores.
`candidate_units`	Character vector of valid VegKey unit codes. `NULL` compares all units.
`use_dominance`	Use dominance constancy for cover above 25% when available.
`output`	Directory for the CSV tables. `NULL` leaves results in R.
`esy_mode`	Run multiple rule files independently with `"separate"` or pool their definitions and priorities with `"merge"`.

The default ESy exclusions `c("7", "8")` refer to JUICE juvenile layers. An Rveg juvenile record (`J`) is mapped to JUICE `7`; Rveg's moss layer (`0`) is mapped to `9`. To include juveniles in ESy, pass `exclude_layers = character(0)`. When training VegKey, specify the exclusions in `CreateVegKey()` so that model building and later classification use the same layers.

7.3 Run the bundled methods on the same plots

Use the installed `example_2` database. The directory in `output` holds the exported files. With both methods requested, the return value has named `esy` and `vegkey` components. A single-method call returns its result tables directly.

```
classification_dir <- file.path(work_dir, "classification")
result <- RvegClassify(
  database = example_db,
  method = c("esy", "vegkey"),
  top_n = 2L,
  output = classification_dir
)
names(result)
#> [1] "esy"    "vegkey"
result$esy$summary
#>   plot_id esy_n_matches esy_code         esy_name
#> 1      X1             1    MCG01 Caricetum elatae
#> 2      X2             1    MCG01 Caricetum elatae
#> 3      X3             0     <NA>             <NA>
result$vegkey$summary[, c("plot_id", "vegkey_code", "n_used_taxa")]
#>   plot_id vegkey_code n_used_taxa
#> 1      X1       LAA02          11
#> 2      X2       LAA02           6
#> 3      X3       LAA02           3
```

In this example ESy assigns `X1` and `X2` to *Caricetum elatae* (`MCG01`). `X3` has no formal match. The bundled VegKey model ranks *Carici elongatae-Alnetum glutinosae* (`LAA02`) first for these plots. Inspect which taxa were used, the other ranked units, and the reference data before interpreting those results.

Join summaries to your header fields with the `X` key. The following uses `match()` to retain the order of the ESy summary.

```
review <- result$esy$summary
review$locality <- plots$HeaderDATA[
  match(review$plot_id, rownames(plots$HeaderDATA)), "LOCALITY"
]
review[, c("plot_id", "locality", "esy_n_matches", "esy_code")]
#>   plot_id locality esy_n_matches esy_code
#> 1      X1       CB             1    MCG01
#> 2      X2       CB             1    MCG01
#> 3      X3       CB             0     <NA>
```

7.3.1 Exported results and taxon diagnostics

`classification.csv` records the plot, method, status, rank, association code and name, score, taxon and training counts when available, and rule source. ESy rows record formal matches and carry no similarity score. FPFI and VegKey rows record ranks and method-specific scores. The export also represents plots with no match or no score so that an absent result can be traced back to a plot. An affected ESy or FPFI row can have status `uncertain` when missing header data affects a formal decision.

`taxa.csv` records the original short code and name, method, matched name, and match status. If more than one ESy file was requested, it also includes rule source. Where an FPFI profile is used, the export can include a separate FPFI mapping because the profile can use names that differ from those in the formal definitions. A taxon can appear in the mapping even when its layer excludes it from a particular score.

```
exported <- read.csv(file.path(classification_dir, "classification.csv"))
taxa <- read.csv(file.path(classification_dir, "taxa.csv"))
exported[exported$plot_id == "X3",
         c("method", "status", "association_code", "score")]
#>   method   status association_code     score
#> 7    esy no_match                        NA
#> 8 vegkey   ranked            LAA02 81.682827
#> 9 vegkey   ranked            LAA01  8.820179
head(taxa[, c("taxon_id", "method", "status")])
#>   taxon_id method    status
#> 1  CARDDEN    esy unmatched
#> 2  CAREELA    esy   matched
#> 3  CAREELO    esy   matched
#> 4  CAREPSU    esy   matched
#> 5  DESHCES    esy   matched
#> 6  DRYOCAR    esy   matched
```

7.4 Read ESy results

For one ESy run, `summary` has one row per selected plot. `esy_n_matches` counts retained definitions at the highest matching priority. The code and name fields are filled only when a unique formal result can be assigned.

esy_n_matches	Interpretation	Where to look next
1	One unit at the highest matching priority.	summary for the code and name.
Greater than 1	Several units tie at the highest priority.	all_matches for all retained codes, and fpfi_ranked if a profile is available.
0	No formal unit was retained.	Check taxa and rules. Use fpfi_ranked for possible similarities when available.
NA	An unknown header value can affect the highest-priority result.	Verify the relevant header field, its units, and the rule.

all_matches retains the top-priority formal results even when several definitions tie. taxon_map links the original codes and names to names used by the rules. ESy rule files can also contain aggregation mappings to interpret records within the system's taxonomic and layer conventions.

7.4.1 FPFI similarity

The full CzechVeg-ESy file includes similarity profiles. The ordinary basic file supplies the formal rules without a usable ranked profile. With a profile, Rveg adds the weights of the matched taxa present in a plot. This gives an FPFI score for each eligible unit. Where a formal match exists, the ranking considers the retained formal units represented in the profile. For a plot with no formal match, it considers the profile's units. A plot with an uncertain formal result is omitted from this ranking.

```
full <- RvegClassify(
  database = example_db, rules = "CzechVeg-ESy-full",
  plot_id = c("X1", "X3"), top_n = 2L
)
full$summary
#>   plot_id esy_n_matches esy_code         esy_name
#> 1      X1             1    MCG01 Caricetum elatae
#> 2      X3             0     <NA>             <NA>
full$fpfi_ranked[, c("plot_id", "rank", "association_code",
                     "fpfi", "n_used_taxa")]
#>   plot_id rank association_code     fpfi n_used_taxa
#> 1      X1    1            MCG01 27.34500          11
#> 2      X3    1            MCG01 16.96827           3
#> 3      X3    2            LAA02  5.85159           3
```

n_used_taxa gives the number of plot taxa represented in the profile used for scoring. Higher scores indicate a closer match by this profile. Check which profiles and taxa went into the result before comparing scores from different setups. Raise min_fpfi to filter low positive scores, or set top_n to see more candidates. A remaining FPFI rank for a plot with zero formal matches is a similarity result. Its formal status stays at zero.

7.5 Read VegKey results

The VegKey model records how frequently each taxon occurs within each reference unit. For a given candidate unit, Rveg combines the constancies of the observed and usable taxa. It uses

logarithms to calculate the product reliably. A constancy floor avoids a zero product from one absent or very infrequent taxon. If dominance data are present and enabled, a taxon exceeding 25% cover uses its dominance constancy for that score.

The result list contains three tables.

Table	Contains
`summary`	Top code and name per plot, its relative score, and number of usable taxa.
`vegkey_ranked`	Rank, unit code and name, relative score, and training plot count where available.
`taxon_map`	Original taxon code and name, and the matching model name where one was found.

```
result$vegkey$vegkey_ranked[, c(
  "plot_id", "rank", "unit_code", "relative_score", "n_training"
)]
#>   plot_id rank unit_code relative_score n_training
#> 1      X1    1     LAA02   1.000000e+02         49
#> 2      X1    2     MCG01   7.019265e-09         43
#> 3      X2    1     LAA02   9.888141e+01         49
#> 4      X2    2     LAA01   1.115592e+00         16
#> 5      X3    1     LAA02   8.168283e+01         49
#> 6      X3    2     LAA01   8.820179e+00         16
```

For intuition, suppose one observed taxon has occurrence constancy 0.80 in unit A and 0.40 in unit B. A second observed taxon has constancy 0.50 in A and 0.80 in B. Under occurrence-only scoring, the products are 0.40 and 0.32. Normalizing the two candidates to sum to 100 gives about 55.6 and 44.4. A change to the candidate set can change the relative scores and the count of plot taxa used, since scoring only uses taxa represented in the selected units.

Scores sum to 100 across the eligible candidate units before `top_n` truncates the table. A high relative score can occur with few usable taxa or with a very narrow candidate pool. Interpret the score together with `n_used_taxa`, `n_training`, the other candidates, and the plot. If the plot has no usable taxa for those units, VegKey returns no rank.

7.6 Diagnose taxon matching

Rveg first tries the original botanical spelling and then a normalized spelling. Normalization covers certain differences such as alternate subspecies abbreviations and hybrid signs. A same-name aggregate can be used where an otherwise unmatched binomial has a corresponding aggregate in the target names. If synonym matching is enabled, the remaining names are checked against the supplied WCVP relationships. Ambiguity can leave a taxon unmatched. Rules and models have different sets of target names, so a successful match for one method need not produce the same mapping for the other.

`synonym_file` choice	How it works
`"WCVP-Czech"`	Uses the bundled Czech table from WCVP v16.

`synonym_file` choice	How it works
`"WCVP-Global"`	Uses the global WCVP v16 table. If it is missing, Rveg offers to download it in an interactive session. Declining uses the Czech table for that run.
A file path	Uses a local UTF-8 TSV, plain or gzip-compressed.
`NULL`	Disables synonym lookup. Exact, normalized, and same-name aggregate matching still apply.

For scripted work, make the global table available before running, or supply a local table explicitly. The download is cached in the Rveg user cache on R 4.0 or newer, and in the session's temporary directory on older R versions. The first download needs a network connection. If an explicitly supplied local file path is missing, the current implementation skips synonym lookup. Verify the file exists for a reproducible run.

A custom table needs columns `accepted_name` and `synonyms`. Use a literal space, pipe, and space (|) between several synonyms. Use one row per accepted name. Supply full names and review the taxonomic meaning of each relationship, particularly for aggregates and infraspecific taxa.

```
read.delim("project_synonyms.tsv", fileEncoding = "UTF-8", quote = "")

use_local_names <- RvegClassify(
  "my_survey",
  method = "vegkey",
  synonym_file = "project_synonyms.tsv"
)
use_local_names$taxon_map
```

Find the taxa unavailable to the bundled model. The bundled example includes two locally added demonstration taxa.

```
subset(result$vegkey$taxon_map, is.na(matched_name))
#>    taxon_id           taxon_name matched_name
#> 21  IMAGSPE Imaginary speciesone         <NA>
#> 22  IMAGSPT Imaginary speciestwo         <NA>
```

For an unexpected match, check the full name in `RvegLoad(variation = 3)$RelDATA`, the database's checklist and any `extra_spec` entry, the taxonomic scope of the chosen model or rule file, and the synonym table used for the run. For a missing score, also check whether the layer was excluded and whether the taxon is represented in the selected units or profile. A target name can be absent even when the name is valid for your region.

8 Write and test custom ESy definitions

8.1 The `CreateESyRules()` arguments

```
CreateESyRules(definitions, file, groups = NULL)
```

Argument	How it is used
`definitions`	A nonempty data frame with `code`, `name`, `formula`, and optional `priority`.
`file`	Destination path. The function writes a UTF-8 rule file and invisibly returns the path.
`groups`	Named list of species vectors. Give each group a distinct name and at least one species.

Put one unit on each row, use a short code without spaces, and choose a name meaningful to the reader. Use a unique code within the file. The priority defaults to `"5"` and accepts a single digit or letter. Higher priorities win in the sequence `0` to `9`, `A` to `Z`, then `a` to `z`. Definitions at the same highest priority can tie. The writer creates the aggregation, species-group, and definition sections. Pass the resulting file to `RvegClassify(rules = ...)`. The generated file has no fitted similarity profiles.

8.2 Write conditions

Use full names and put each predicate inside angle brackets. Write `AND`, `OR`, and `NOT` in uppercase. The ESy syntax `A NOT B` means `A AND (not B)`. Parentheses are useful when combining disjunctions and exclusions. Species cover thresholds use percentage values.

Example	Meaning
`<Carex elata GR 30>`	*Carex elata* cover is strictly above 30%. `UP` also expresses a strict greater-than comparison.
`<Carex elata GE 30>`	Cover is at least 30%.
`<Carex elata EQ 0>`	Cover is exactly zero in a plot evaluated by the rule.
`<$$N PlotSize GE 16>`	The numeric header field `PlotSize` is at least 16.
`<$$C CRS EQ WGS84>`	The character header field `CRS` has the value `WGS84`.

You can write `<#TC WetSedges GE 25>` to test the combined cover of a named group. `#TC` applies probabilistic cover aggregation, while `#SC` uses the highest individual cover in the group. A `###` group can express a species-count condition. A bare group reference such as `<### WetSedges>` requires both taxa in a two-species group, or at least half rounded up for larger groups. If you supply only the group name, the file writer adds `###`. You can include the group type in the name, as in `"#TC WetSedges"`, in the `groups` argument. Reference the same name in the formula.

Taxon cover here refers to the records remaining after layer exclusions and the rule file's aggregation. Formal rules are evaluated for plots with usable records after that preparation. A cover equality such as `EQ 0` therefore does not turn an empty plot into a formal match. For a header condition, ensure the required field exists and its value is recorded consistently. A blank or missing value can leave a higher-priority rule undecidable.

8.3 Build a small rule set and inspect matches

The following illustrative definitions use the bundled wetland example. The high-priority unit requires both *Carex elata* cover above 30% and the group's cover of at least 35%. A more general unit accepts either *Carex elata* or *Phragmites australis* above 5%. These examples demonstrate the syntax using the installed data.

```
wetland_rules <- data.frame(
  code = c("SEDGE", "WET"),
  name = c("Tall-sedge stand", "General wetland stand"),
  formula = c(
    "<Carex elata GR 30> AND <#TC WetSedges GE 35>",
    "<Carex elata GR 5> OR <Phragmites australis GR 5>"
  ),
  priority = c("7", "5")
)
rule_file <- file.path(work_dir, "wetland_rules.txt")
CreateESyRules(
  definitions = wetland_rules, file = rule_file,
  groups = list(WetSedges = c("Carex elata", "Carex acutiformis"))
)

wetland_esy <- RvegClassify(example_db, rules = rule_file)
wetland_esy$summary
#>   plot_id esy_n_matches esy_code              esy_name
#> 1      X1             1    SEDGE      Tall-sedge stand
#> 2      X2             1    SEDGE      Tall-sedge stand
#> 3      X3             1      WET General wetland stand
wetland_esy$all_matches[, c("plot_id", "association_code")]
#>   plot_id association_code
#> 1      X1            SEDGE
#> 2      X2            SEDGE
#> 3      X3              WET
```

Inspect the UTF-8 file with `readLines(rule_file, encoding = "UTF-8")` to see its sections and prioritised labels. When changing a threshold, test plots near the boundary and check whether you intended a strict or inclusive comparison. To see all ties within a priority, inspect `all_matches` for the plot of interest.

8.3.1 Compare independent rule files and shared priorities

Run several rule files independently when you want to review each system's result. This is the default. To have their definitions compete at a shared priority, set `esy_mode = "merge"`. Review the priority conventions in each input before combining them. Use `rule_source` to trace the unit back to its file.

```
second_file <- file.path(work_dir, "reed_rules.txt")
CreateESyRules(
  definitions = data.frame(
    code = "REED", name = "Reed stand", priority = "8",
    formula = "<Phragmites australis GR 5>"
```

```
  ),
  file = second_file
)

files <- c(rule_file, second_file)
separate <- RvegClassify(example_db, plot_id = "X1", rules = files)
pooled <- RvegClassify(
  example_db, plot_id = "X1", rules = files, esy_mode = "merge"
)

separate$summary[, c("plot_id", "esy_code", "rule_source")]
#>   plot_id esy_code       rule_source
#> 1      X1    SEDGE wetland_rules.txt
#> 2      X1     REED    reed_rules.txt
pooled$summary[, c("plot_id", "esy_code", "rule_source")]
#>   plot_id esy_code    rule_source
#> 1      X1     REED reed_rules.txt
```

You can also include `"CzechVeg-ESy-basic"` in a vector of local rule paths. Separate mode returns one summary row for each plot and file. Merged mode returns one summary row per plot and applies priority across the combined definitions. Here `X1` meets both definitions. The reed definition has priority 8, so it wins in merged mode. When reviewing merged ties, use the combination of source and association code to identify the original definition.

8.3.2 Check a header-dependent rule

The example deliberately refers to a header the bundled database does not contain. The plot then lacks a numeric value needed to decide a higher-priority rule. Supply a valid field in your database before using such a rule operationally.

```
header_rules <- data.frame(
  code = c("WET", "LARGE"),
  name = c("Wetland stand", "Large wetland plot"),
  formula = c(
    "<Carex elata GR 5>",
    "<Carex elata GR 5> AND <$$N SurveyArea GE 16>"
  ),
  priority = c("5", "8")
)
header_file <- file.path(work_dir, "header_rules.txt")
CreateESyRules(header_rules, file = header_file)
header_result <- RvegClassify(
  example_db, rules = header_file, plot_id = "X1"
)
header_result$summary
#>   plot_id esy_n_matches esy_code esy_name
#> 1      X1            NA     <NA>     <NA>
```

To resolve this case, use the actual field name and units (for example, `PlotSize` in square metres) and fill any missing values from the original sheets. Run the revised rule again and compare the resulting match counts.

9 Build and use a VegKey model

9.1 The `CreateVegKey()` arguments

```
CreateVegKey(
  database = NULL, unit_column = NULL, synoptic = NULL, file,
  unit_name_column = NULL, checklist = "default", customscale = FALSE,
  exclude_layers = c("7", "8"), source = NULL, min_constancy = 0.0001,
  constancy_scale = c("percent", "proportion")
)
```

Provide either classified reference relevés via `database` or an existing synoptic table via `synoptic`. Both routes write an `RvegVegKey` object to the `.rds` path and return that path invisibly. Rveg saves this object using R serialization version 2 for compatibility with older versions of R.

Argument	How it is used
`database`	Name or path of the reference database, or a list of `headers` and `species` tables with plot keys.
`unit_column`	Required for the reference route. Field holding the reviewed unit codes.
`synoptic`	Data frame or CSV path with one row per taxon and unit.
`file`	Output filename ending in `.rds` in an existing directory.
`unit_name_column`	Optional reference header field holding the unit names. Where unavailable, codes are used as names.
`checklist`, `customscale`	Passed to `RvegLoad()` for the on-disk reference database route.
`exclude_layers`	JUICE layer codes omitted from reference preparation and stored for later prediction.
`source`	Optional provenance, for example the name and version of the reference data.
`min_constancy`	Positive scoring floor no greater than 1. The default is 0.0001 on the proportion scale.
`constancy_scale`	Synoptic columns are percentages by default. Use `"proportion"` for values between 0 and 1.

9.2 Train from labelled relevés

For an existing Rveg database, add a header field such as `VegetationUnit` and fill it with the reviewed classification. A second field can store the readable name. Decide which layers and cover conversions correspond to the ecological question. The layer settings must also make sense for the plots to be classified.

```
reference_model <- CreateVegKey(
  database = "wetland_reference",
  unit_column = "VegetationUnit",
  unit_name_column = "UnitName",
  file = "wetland_reference.rds",
  source = "Reviewed wetland reference plots, 2026"
)

RvegClassify("new_wetland_plots", method = "vegkey", model = reference_model)
```

A plot contributes to the denominator of its unit when it has a nonblank unit code and at least one usable taxon after layer exclusions. Occurrence constancy is the number of contributing plots in that unit where the taxon occurs, divided by the total number of contributing plots. Records of the same taxon from multiple included layers are combined into one cover per plot. When all usable cover records are known, the dominance constancy for a taxon is estimated as the fraction of contributing plots where its combined cover exceeds 25%.

9.2.1 Train from reference tables already in R

For tabular reference data, supply a list with `headers` and `species` data frames. Use one header record per plot and join the tables by `plot_id`. The required species columns are `plot_id`, `taxon_name`, `layer`, and `cover_pct`. For this route, `layer` uses JUICE codes such as `"6"` for herbs. Use full taxon names and numeric percentage cover. An unknown cover can be `NA_real_` for an observed taxon. If a usable cover is unknown, the model uses occurrence constancy without estimating dominance.

```
reference <- list(
  headers = data.frame(
    plot_id = c("p1", "p2", "p3", "p4"),
    unit = c("A1", "A1", "A2", "A2"),
    label = c("Sedge stand", "Sedge stand", "Reed stand", "Reed stand")
  ),
  species = data.frame(
    plot_id = c("p1", "p1", "p2", "p3", "p4", "p4"),
    taxon_name = c(
      "Carex elata", "Phragmites australis", "Carex elata",
      "Phragmites australis", "Carex elata", "Phragmites australis"
    ),
    layer = "6",
    cover_pct = c(55, 8, 30, 70, 7, 45)
  )
)

reference_file <- file.path(work_dir, "reference_model.rds")
CreateVegKey(
  database = reference, unit_column = "unit",
  unit_name_column = "label", file = reference_file,
  source = "Four illustrative reference plots"
)
fitted <- readRDS(reference_file)
```

Inspect the reference counts and the two constancy matrices. Each matrix has taxa in rows and vegetation units in columns.

```
fitted$units
#>   code        name n_plots
#> 1   A1 Sedge stand       2
#> 2   A2  Reed stand       2
fitted$presence
#>                      A1  A2
#> Carex elata          1.0 0.5
```

```
#> Phragmites australis 0.5 1.0
fitted$dominant
#>                       A1 A2
#> Carex elata            1  0
#> Phragmites australis   0  1
```

In unit A1, *Carex elata* is present and exceeds 25% in both training plots. *Phragmites australis* is present in one of the two A1 plots and exceeds 25% in neither. This yields occurrence constancies of 1 and 0.5 and dominance constancies of 1 and 0, respectively. The two plots per unit in this illustration make the calculation easy to trace. For a study, review reference coverage and evaluate the model on independent plots.

9.3 Train from a synoptic table

The synoptic route uses existing taxon frequencies. Supply a data frame or a UTF-8 comma-delimited CSV with one row per unit and taxon. Use consistent codes, one agreed name per unit, and one row per unit–taxon pair. Omitted combinations enter the model at zero constancy. A taxon never present in any unit is excluded from scoring.

Column	Meaning
`unit_code`	Required. Unit identifier used by `candidate_units`.
`taxon_name`	Required. Botanical name for matching query taxa.
`constancy`	Required. Occurrence constancy of the taxon in the unit.
`unit_name`	Optional. Readable unit name.
`n_plots`	Optional. Number of reference plots for the unit. Use a consistent count for all rows of the same unit.
`dominant_constancy`	Optional. Fraction or percentage of plots in the unit where the taxon exceeds 25% cover. Use the same scale as `constancy`.

Use values from 0 to 100 with the default `"percent"`, or 0 to 1 with `constancy_scale = "proportion"`. For example, 85% is written as `85` in the first case and `0.85` in the second. The illustration gives the same number of plots in the rows belonging to each unit.

```
synoptic <- data.frame(
  unit_code = c("A1", "A1", "A2", "A2"),
  unit_name = rep(c("Sedge stand", "Reed stand"), each = 2),
  taxon_name = rep(c("Carex elata", "Phragmites australis"), 2),
  constancy = c(95, 10, 10, 95),
  dominant_constancy = c(60, 2, 2, 65),
  n_plots = c(40L, 40L, 35L, 35L)
)

synoptic_file <- file.path(work_dir, "wetland_synoptic.csv")
write.csv(synoptic, synoptic_file, row.names = FALSE)
model_file <- file.path(work_dir, "wetland_model.rds")
CreateVegKey(synoptic = synoptic_file, file = model_file,
             source = "Illustrative wetland synoptic table")
```

9.4 Classify with your model and check the result

Use the resulting file as `model`, or pass the object returned by `readRDS()`. Here the model contains two units. The first call compares both. The second restricts candidates and uses occurrence constancy only.

```
wetland <- RvegClassify(
  example_db, method = "vegkey", model = model_file,
  plot_id = c("X1", "X3")
)
wetland$summary[, c("plot_id", "vegkey_code", "n_used_taxa")]
#>   plot_id vegkey_code n_used_taxa
#> 1      X1          A1           2
#> 2      X3          A1           1
wetland$vegkey_ranked[, c("plot_id", "rank", "unit_code",
                          "relative_score", "n_training")]
#>   plot_id rank unit_code relative_score n_training
#> 1      X1    1        A1       75.94937         40
#> 2      X1    2        A2       24.05063         35
#> 3      X3    1        A1       90.47619         40
#> 4      X3    2        A2        9.52381         35

occurrence_only <- RvegClassify(
  example_db, method = "vegkey", model = readRDS(model_file),
  candidate_units = "A1", use_dominance = FALSE, plot_id = "X1"
)
occurrence_only$summary
#>   plot_id n_used_taxa vegkey_code vegkey_name relative_score
#> 1      X1           2          A1 Sedge stand            100
```

With one eligible candidate and usable taxa, the relative score is 100 by construction. Read `n_used_taxa` before drawing conclusions from a narrow candidate list. For model review, compare predictions with independently classified plots, inspect less common units, and check that the training and target datasets use compatible layer, cover, and nomenclatural conventions. Record which reference data, model, checklist, and synonym table produced an exported result.

10 Troubleshooting and project resources

Symptom	What to check
The database cannot be opened.	Give the database name or path without `HEAD.csv` or `REL.csv`. Check that the two files are together and their plot columns agree.
`Y` or `YSP` reports different header and relevé counts.	Review `PRINTHEAD` and `PRINTREL`, then complete the missing columns in the correct order.
A short code has no botanical name.	Use the checklist recorded for the project and check `extra_spec`. Verify the seven-character code and layer suffix in the raw data.
A cover table contains character values.	Inspect the stored symbols and convert the original custom scale where appropriate.

Symptom	What to check
The merge reports a code collision.	Read the reported mapping. Review the target checklist and any new extra-species entries against the original plot records.
A Turboveg CSV import fails where an XML import works.	Compare the combined layout and separators with the bundled spreadsheet. Confirm the CSV encoding and the position of the header/species separator.
ESy returns `NA` in `esy_n_matches`.	Check any header variable used at an equal or higher priority and confirm its type and units.
An FPFI table is empty.	Check whether the rule file has a usable profile, whether profile taxa occur after the layer exclusions, and whether the score threshold filters all candidates.
VegKey returns no unit.	Check model names, excluded layers, and taxa represented in the candidate units.
Global WCVP synonyms are unavailable in a script.	Download once interactively in the same environment or supply a local table.

For package development, release notes, or to report a problem, use the Rveg GitHub repository and its issue tracker. Include the Rveg version, the call and console output, and a small example that reproduces the problem. The Plant Ecology Lab project page links to project resources.

11 References

- Král, P. and Douda, J. (2025). Rveg methodology. https://doi.org/10.1111/jvs.70012
- Danihelka, J., Chrtek, J. and Kaplan, Z. (2012). Checklist of vascular plants of the Czech Republic. *Preslia* 84, 647–811. https://www.preslia.cz/article/159
- Hennekens, S. M. and Schaminée, J. H. J. (2001). TURBOVEG, a comprehensive data base management system for vegetation data. https://doi.org/10.2307/3237010
- Tichý, L. (2002). JUICE, software for vegetation classification. https://doi.org/10.1111/j.1654-1103.2002.tb02069.x
- Chytrý, M. et al. (2020). CzechVeg-ESy. https://doi.org/10.5281/zenodo.3605562
- Tichý, L. and Chytrý, M. (2019). VegKey. https://doi.org/10.1111/jvs.12799
- Chytrý, M. et al. (2026). CzechVeg-OpenStrat. https://doi.org/10.5281/zenodo.19231152
- Govaerts, R. (2026). World Checklist of Vascular Plants, version 16, the taxonomic backbone of Plants of the World Online. https://doi.org/10.15468/6h8ucr
- Tichý, L., Holt, J. and Nejezchlebová, M. (2011). JUICE Program Manual, 2nd edition. https://www.sci.muni.cz/botany/juice/?idm=9