

# Input/output

Kenneth B. Hoehn

2026-08-19

## Contents

|                                                                       |   |
|-----------------------------------------------------------------------|---|
| Reading AIRR Rearrangement sequencing data files . . . . .            | 1 |
| scRepertoire and Dowser compatability . . . . .                       | 2 |
| Saving and loading Dowser objects in AIRR Clone JSON format . . . . . | 2 |
| Saving and loading Dowser objects as RDS . . . . .                    | 3 |
| Exporting trees in Newick format . . . . .                            | 3 |
| Exporting and reading sequences as FASTA . . . . .                    | 3 |
| Making and saving tree plots . . . . .                                | 4 |

This vignette covers how data gets into Dowser and the different ways Dowser objects and trees can be saved. These formats primarily follow the Adaptive Immune Receptor Repertoire (AIRR) Community standard formats. Specifically this will cover AIRR TSV input, as well as AIRR Trees and Clones JSON format (`writeTreesJSON/readTreesJSON`), plain R serialization (`saveRDS/readRDS`), Newick tree export (`exportTrees`), and FASTA sequence export/import (`writeFasta/readFasta`).

## Reading AIRR Rearrangement sequencing data files

The best input format for Dowser is the AIRR Rearrangement TSV format. The `airr` package's `read_rearrangement` function reads such a file into a data frame:

```
library(airr)
library(dowser)

# Read in an AIRR-formatted TSV file of Ig/TCR rearrangements
airr_data <- read_rearrangement("sequences.tsv")
```

To keep this vignette self-contained, we'll instead use the `ExampleAirr` data object included with Dowser, which is the kind of data frame `read_rearrangement` would return.

```
library(airr)
library(dowser)

# load example AIRR data, as if read in with read_rearrangement
data(ExampleAirr)
```

```
# subset data for this example
ExampleAirr <- ExampleAirr[ExampleAirr$clone_id %in% c("3170", "3184"),]
```

From here, `formatClones` groups sequences into clones and reconstructs germlines, and `getTrees` builds a lineage tree for each clone (see the Build Lineage Trees vignette for details on both steps).

Before this step, it's important that the BCR sequences are already grouped into clones beforehand (they will have a `clone_id` column). This can be accomplished using SCOPer. Further, it is important that each clonal germline V/J has been reconstructed, which can be accomplished using `(createGermlines)[https://dowser.readthedocs.io/en/stable/vignettes/Germlines-Vignette/]`. This will generate a `germline_alignment_d_mask` column. To see all of these steps together, check out one of the full Immcantation tutorials.

```
# Process example data into proper format, store isotype (optional)
clones <- formatClones(ExampleAirr, traits="c_call")
```

```
# Build maximum parsimony trees for each clone
trees <- getTrees(clones, nproc=1)
```

```
print(trees)
```

`trees` is a standard Dowser object: a tibble with one row per clone with an `airrClone` object in the `data` column and a tree (`ape::phylo`, or `treeio::treedata` for time trees) in the `trees` column. The rest of this vignette covers ways to save and reload this object.

## scRepertoire and Dowser compatability

While Dowser was designed to work downstream of Immcantation packages, it is possible to use Dowser and other Immcantation packages in combination with scRepertoire.

To learn how to do this, check out the scRepertoire team's excellent tutorial about Combining Immcantation and scRepertoire.

## Saving and loading Dowser objects in AIRR Clone JSON format

To save an entire Dowser object, including trees, use `writeTreesJSON` to write a Dowser object to an AIRR Clone JSON file. Conversely, `readTreesJSON` reads it back. This works for trees built with any `getTrees` build option, as well as time trees from `getTimeTrees/` `getTimeTreesIterate`. By default, `writeTreesJSON` reloads the file it just wrote and checks that it matches the original object before returning.

```
# Write tree object to a JSON file, checking it reads back identically
writeTreesJSON(trees, "trees.json")
```

```
# Read it back in
trees_json <- readTreesJSON("trees.json")
```

```
print(trees_json)
```

Because it's plain JSON with a documented schema, this format is useful for archiving trees long-term, sharing them with collaborators who aren't using R, or loading them into other tools.

**Note** that internal node labels are not preserved when inputting/outputting in this schema, but this is only relevant if you're manually selecting particular nodes, such as reconstructing internal node sequences. All other aspects of the Dowser object are preserved and checked on output.

## Saving and loading Dowser objects as RDS

If you're staying within R, `saveRDS/readRDS` is the simplest way to save a Dowser object exactly as-is, with no format translation, although it is a binary file that is unreadable outside of R.

```
# Save tree object to an RDS file
saveRDS(trees, "trees.rds")
```

```
# Read it back in
trees_rds <- readRDS("trees.rds")
```

```
print(trees_rds)
```

## Exporting trees in Newick format

`exportTrees` writes the trees in a Dowser object in order to a single Newick file, one tree per line, using `ape::write.tree`.

Alternatively, the list of tree objects in the `trees` column can be exported using many other output functions, for example `ape::write.tree`.

```
# Export trees to a Newick file
exportTrees(trees, "trees.newick")
```

```
# write a single tree
ape::write.tree(trees$trees[[1]], "clone1.tree")
```

## Exporting and reading sequences as FASTA

`writeFasta` writes a named list of sequences to a FASTA file, and `readFasta` reads a FASTA file back into a named list. These are generic sequence I/O functions and provided for convenience.

```
# Get sequences and sequence IDs for the first clone
seqs <- clones$data[[1]]@data$sequence
names(seqs) <- clones$data[[1]]@data$sequence_id
```

```
# Write them to a FASTA file
writeFasta(seqs, "clone_3170.fasta")
```

```
# Read them back in
seqs_read <- readFasta("clone_3170.fasta")
```

```
print(names(seqs_read))
```

Alternatively, `dfToFasta` writes sequences straight from a data frame or tibble to a FASTA file, without needing to build a named list first. By default it reads sequence IDs and sequences from the `sequence_id/sequence` columns, strips IMGT gap characters (`.`), and can append extra columns to each sequence's header.

```
# Get the data frame of sequences for the first clone
```

```
df <- clones$data[[1]]@data
```

```
# Write to FASTA, tagging each header with its isotype
```

```
dfToFasta(df, "clone_3170_df.fasta", columns="c_call")
```

```
cat(readLines("clone_3170_df.fasta")[1:2], sep="\n")
```

Use `id/seq` to point at differently-named columns, and `imgt_gaps=TRUE` to keep IMGT gaps rather than stripping them.

## Making and saving tree plots

Now that trees are built and saved, see the Plotting Trees vignette for how to visualize trees and save tree plots as files.