airr-standards Documentation

Release 1.3

AIRR Community

Jun 01, 2020

Contents

The Adaptive Immune Receptor Repertoire (AIRR) Community of The Antibody Society is a research-driven group that is organizing and coordinating stakeholders in the use of next-generation sequencing (NGS) technologies to study antibody/B-cell and T-cell receptor repertoires. Recent advances in sequencing technology have made it possible to sample the immune repertoire in exquisite detail. AIRR sequencing has enormous promise for understanding the dynamics of the immune repertoire in vaccinology, infectious disease, autoimmunity, and cancer biology, but also poses substantial challenges. The AIRR Community was established to meet these challenges.

CHAPTER 1

Introduction to the AIRR Standards

The AIRR Community is developing a set of standards for describing, reporting, storing, and sharing adaptive immune receptor repertoire (AIRR) data, such as sequences of antibodies and T cell receptors (TCRs). Some specific efforts include:

- The MiAIRR standard for describing minimal information about AIRR datasets, including sample collection and data processing information.
- Data submission guidelines and workflows.
- Data representations (file format) specifications for storing large amounts of annotated AIRR data.
- API to query and download AIRR data from repositories/databases as part of the AIRR Data Commons.
- A community standard for software tools which will allow conforming tools to gain community recognition.
- Set of reference software tools for reading, writing and validating data in the AIRR standards.
- A database and web submission frontend for inferred germline genes

CHAPTER 2

Table of Contents

2.1 Getting Started

This website provides information and resources regarding the AIRR Community Standards for the diverse community of immunology researchers, bioinformaticians, and software developers.

2.1.1 MiAIRR standard for study data submission

- Gather experimental and analysis information about your study to conform to the *MiAIRR* standard (minimal information about adaptive immune receptor repertoires).
- *Submission* of your study data to a public repository.

2.1.2 AIRR Data Commons for query and download of AIRR-seq data

- *Query* publicly available AIRR-seq studies in the *AIRR Data Commons*.

2.1.3 Resources related to data representations and software development

- Schema, definitions and file formats for the *AIRR Data Model*. The AIRR Data Model defines the structure and relationship for the MiAIRR data elements.
- *Software guidelines* for tools developers to enable rigorous and reproducible immune repertoire research.
- *AIRR Data Commons API* provides programmatic access to query and download AIRR-seq data.

2.1.4 Software tools and libraries

- *Python reference library* for reading/writing/validating AIRR data files.
- *R reference library* for reading/writing/validating AIRR data files.

- *ADC API reference implementation* for a local data repository.
- *Resources and tools* that support the AIRR Standards.

2.1.5 Tutorials, examples and workflows

AIRR Rearrangement TSV Interoperability Example

The example that follows illustrates the interoperability provided by the AIRR Rearrangement schema. The code provided demonstrates how to take AIRR formatted data output by IgBLAST and combine it with data processed by IMGT/HighV-QUEST that has converted to the AIRR format by Change-O. Then, the merged output of these two distinct tools is used to (a) create MiAIRR compliant GenBank/TLS submission files, and (b) perform a simple V gene usage analysis task.

Data

We've hosted a small set of example data from BioProject PRJNA338795 (Vander Heiden et al, 2017. J Immunol.) containing both input and output of the example. It may be downloaded from:

Example Data

Walkthrough

Environment setup

We'll use the Immcantation docker image for this example, which comes loaded with all the tools used in the steps that follow:

```
# Download the image
docker pull kleinstein/immcantation:devel

# Invoke a shell session inside the Immcantation docker image
# Map example data (~/data) to the container's /data directory
$> docker run -it -v ~/data:/data:z kleinstein/immcantation:devel bash
```

Generate AIRR formatted TSV files

TSV files compliant with the AIRR Rearrangement schema may be output directly from IgBLAST v1.9+ or generated from IMGT/HighV-QUEST output (or IgBLAST <=1.8 ouput) using the MakeDb parser provided by Change-O:

```
# Generate TSV directly with IgBLAST
$> cd /data
$> export IGDATA=/usr/local/share/igblast
$> igblastn -query HD13M.fasta -out HD13M_fmt19.tsv -outfmt 19 \
      -germline_db_V $IGDATA/database/imgt_human_ig_v \
      -germline_db_D $IGDATA/database/imgt_human_ig_d \
      -germline_db_J $IGDATA/database/imgt_human_ig_j \
      -auxiliary_data $IGDATA/optional_file/human_gl.aux \
      -ig_seqtype Ig -organism human \
      -domain_system imgt
```

(continues on next page)

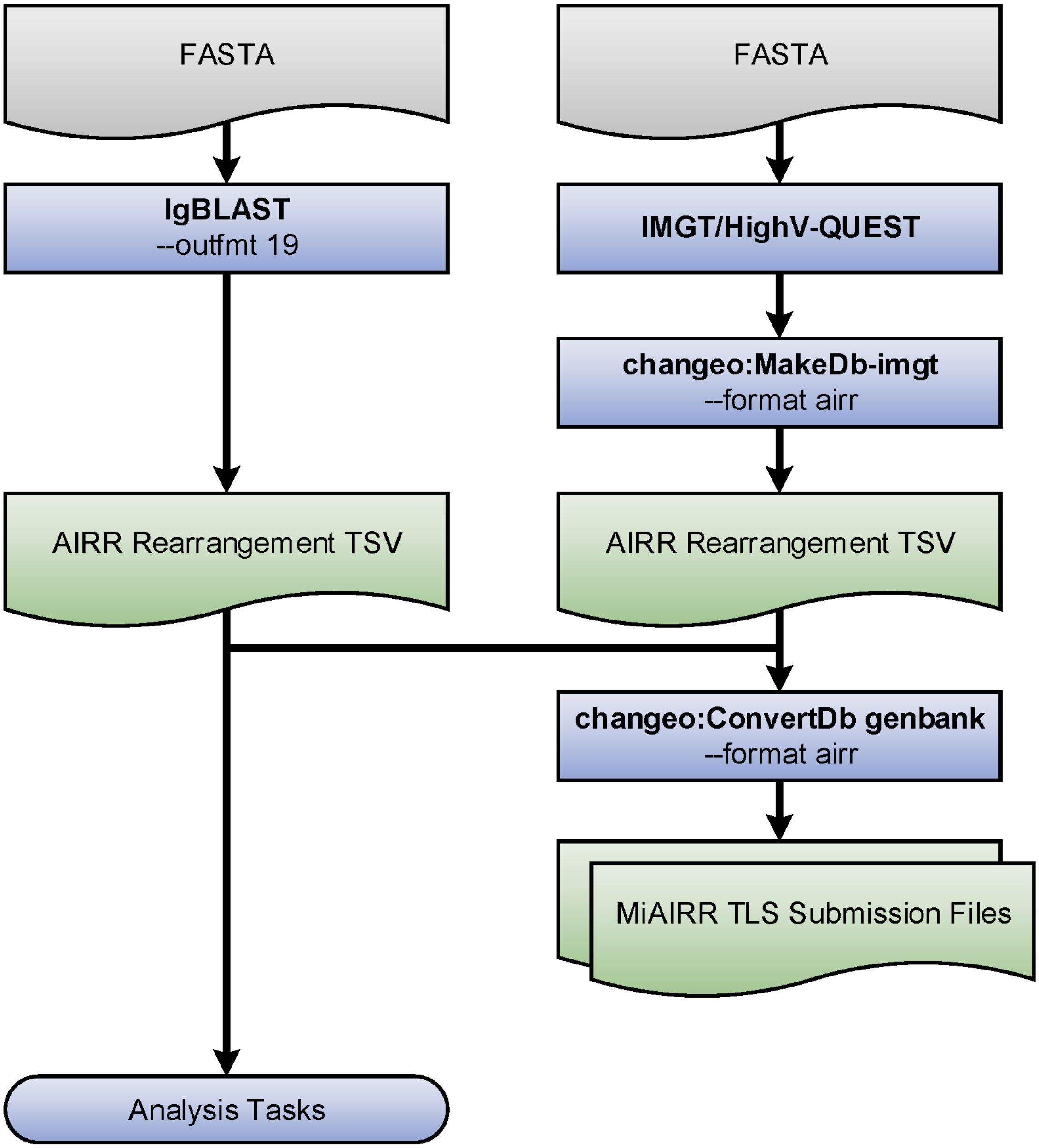


Fig. 1: **Flowchart of the example steps.**

(continued from previous page)

```
# Generate TSV from IMGT/HighV-QUEST results using changeo:MakeDb
$> MakeDb.py imgt -i HD13N_imgt.txz -s HD13N.fasta  \
      --scores --partial --format airr
```

Generate GenBank/TLS submission files

AIRR TSV files can be input directly in Change-O's ConvertDb-genbank tool to generate MiAIRR compliant files for submission to GenBank/TLS:

```
# Generate ASN files from IgBLAST output
$> ConvertDb.py genbank -d HD13M_fmt7_db-pass.tsv --format airr \
      --inf IgBLAST:1.7.0 --organism "Homo sapiens" \
      --tissue "Peripheral blood" --cell "naive B cell" \
      --id --asn -sbt HD13M.sbt

# Generate ASN files from IMGT/HighV-QUEST output
$> ConvertDb.py genbank -d HD13N_imgt_db-pass.tsv --format airr \
      --inf IMGT/HighV-QUEST:1.5.7.1 --organism "Homo sapiens" \
      --tissue "peripheral blood" --cell "naive B cell" \
      --cregion c_call --id --asn -sbt HD13M.sbt
```

Merge files and count V family usage

AIRR TSV files from different tools and easy combined to perform analysis on data generated using different software. Below is shown a simple V family usage analysis after merging the IgBLAST and IMGT/HighV-QUEST outputs into a single table:

```
# Count V family usage in R
# Imports
$> R
R> library(alakazam)
R> library(dplyr)
R> library(ggplot2)

# Merge IgBLAST and IMGT/HighV-QUEST results
R> db_m <- read.delim("HD13M_fmt7_db-pass.tsv")
R> db_n <- read.delim("HD13N_imgt_db-pass.tsv")
R> db_m$cell_type <- "memory"
R> db_n$cell_type <- "naive"
R> db <- bind_rows(db_m, db_n)

# Subset to heavy chain
R> db <- subset(db, grepl("IGH", v_call))

# Count combined V gene usage
R> v_usage <- countGenes(db, "v_call", groups="cell_type",
                         mode="family")

# Plot V family usage
R> ggplot(v_usage, aes(x=GENE, y=SEQ_FREQ, fill=cell_type)) +
    geom_col(position="dodge") +
    scale_fill_brewer(name="Cell type", palette="Set1") +
```

(continues on next page)

(continued from previous page)

```
    xlab("") +
    ylab("Fraction of repertoire")
```

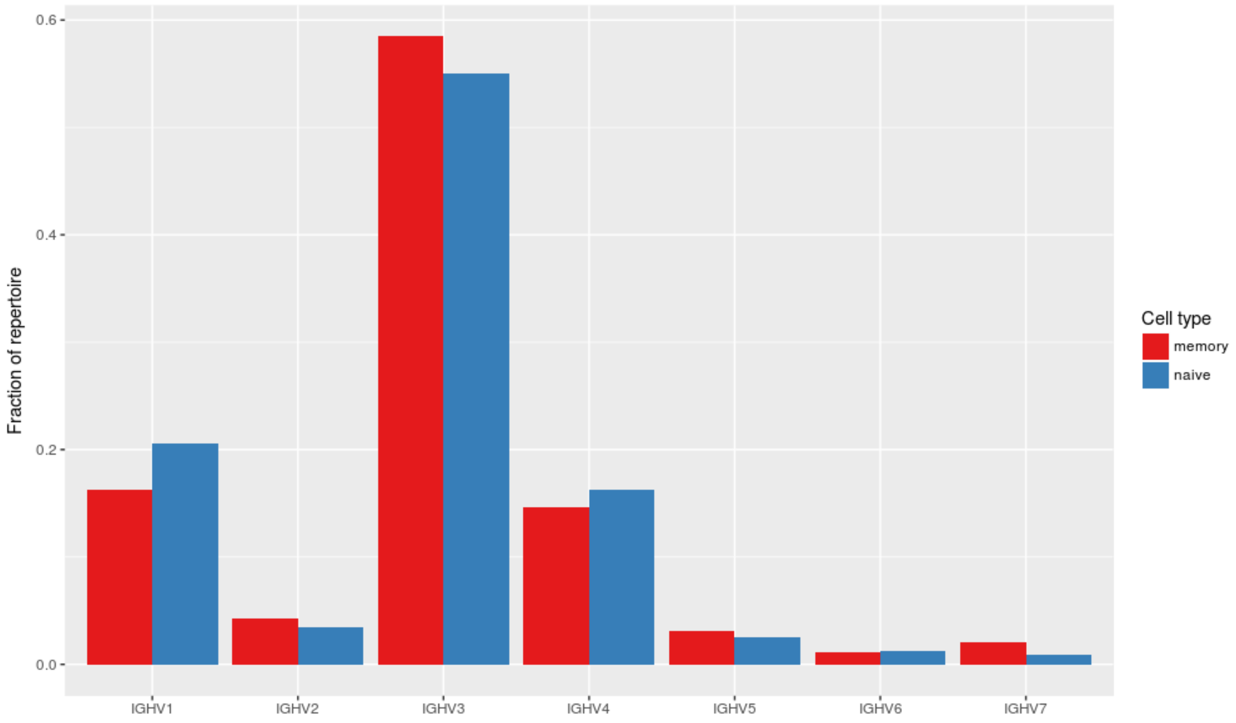


Fig. 2: **V family usage for the combined data set.**

ADC API Query and Analysis Example

This example shows how repertoires and associated rearrangments may be queried from a data repository using the ADC API and then a simple analysis is performed. The example is split between two python scripts; one that performs the query and saves the data into files, and another that reads the data from the files and generates a grouped CDR3 amino acid length distribution plot. The two scripts could be combined into one, but this example illustrates how the data can be saved into files for later use. The example uses the AIRR standards python library.

Data

This example retrieves data for the following study, which is identified by NCBI BioProject PRJNA300878. In this example, we are only going to query and retrieve the T cell repertoires.

Rubelt, F. et al., 2016. Individual heritable differences result in unique cell lymphocyte receptor repertoires of naive and antigen-experienced cells. Nature communications, 7, p.11112.

Basic study description:

- 5 pairs of human twins
- B-cells and T-cells sequenced
- B-cells sorted into naive and memory

- T-cells sorted into naive CD4, naive CD8, memory CD4 and memory CD8
- Total of 60 repertoires: 20 B-cell repertoires and 40 T-cell repertoires

Walkthrough

We'll use the airr-standards docker image for this example, which comes loaded with all the python packages needed. You will want to map a local directory inside the docker container so you can access the data and analysis results afterwards:

```
# Download the image
docker pull airrc/airr-standards:latest

# Make local temporary directory to hold the data
mkdir adc_example
cd adc_example

# Invoke a shell session inside the docker image
docker run -it -v $PWD:/data airrc/airr-standards:latest bash
```

The first python script queries the data from the VDJServer data repository and saves them into files:

```
# Query the data
cd /data
python3 /airr-standards/docs/examples/api/retrieve_data.py
```

Only a subset of the data is downloaded for illustration purposes, but review the code to see how all data can be downloaded. A total of 40 repertoires and 300,178 rearrangements should be downloaded. The repertoire metadata is saved in the `repertoires.airr.json` file, and the rearrangements are saved in the `rearrangements.tsv` file. The script should take a few minutes to run and produce the following display messages:

```
       Info: VDJServer Community Data Portal
    version: 1.3
description: VDJServer ADC API response for repertoire query
Received 40 repertoires.
Retrieving rearrangements for repertoire: 5168912186246295065-242ac11c-0001-012
Retrieved 9768 rearrangements for repertoire: 5168912186246295065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 5338391595746455065-242ac11c-0001-012
Retrieved 5521 rearrangements for repertoire: 5338391595746455065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 4858300151399575065-242ac11c-0001-012
Retrieved 2885 rearrangements for repertoire: 4858300151399575065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 5039977268020375065-242ac11c-0001-012
Retrieved 4053 rearrangements for repertoire: 5039977268020375065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6240077029868695065-242ac11c-0001-012
Retrieved 3506 rearrangements for repertoire: 6240077029868695065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6389112395039895065-242ac11c-0001-012
Retrieved 2289 rearrangements for repertoire: 6389112395039895065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 5939858815878295065-242ac11c-0001-012
Retrieved 3637 rearrangements for repertoire: 5939858815878295065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6088937130722455065-242ac11c-0001-012
Retrieved 9028 rearrangements for repertoire: 6088937130722455065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7446748091679895065-242ac11c-0001-012
Retrieved 1540 rearrangements for repertoire: 7446748091679895065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7591789137265815065-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 7591789137265815065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7066128089908375065-242ac11c-0001-012
```

(continues on next page)

(continued from previous page)

```
Retrieved 5662 rearrangements for repertoire: 7066128089908375065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 5624006920930455065-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 5624006920930455065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8961797805343895065-242ac11c-0001-012
Retrieved 1179 rearrangements for repertoire: 8961797805343895065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 9084118473933975065-242ac11c-0001-012
Retrieved 4464 rearrangements for repertoire: 9084118473933975065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8485700680582295065-242ac11c-0001-012
Retrieved 3908 rearrangements for repertoire: 8485700680582295065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7309695685264535065-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 7309695685264535065-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8425807333172056551-242ac11c-0001-012
Retrieved 6863 rearrangements for repertoire: 8425807333172056551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8263242821018456551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 8263242821018456551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8733756488295256551-242ac11c-0001-012
Retrieved 5298 rearrangements for repertoire: 8733756488295256551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 8602072790999896551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 8602072790999896551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7313153105470296551-242ac11c-0001-012
Retrieved 9121 rearrangements for repertoire: 7313153105470296551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6964444710708056551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 6964444710708056551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7640859110155096551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 7640859110155096551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7461458326201176551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 7461458326201176551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 5953881855632216551-242ac11c-0001-012
Retrieved 5916 rearrangements for repertoire: 5953881855632216551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 7158276584776536551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 7158276584776536551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6393557657723736551-242ac11c-0001-012
Retrieved 7257 rearrangements for repertoire: 6393557657723736551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 6205695788196696551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 6205695788196696551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 4476756703191896551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 4476756703191896551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 4357957907784536551-242ac11c-0001-012
Retrieved 7033 rearrangements for repertoire: 4357957907784536551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 4931851437876056551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 4931851437876056551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 4744762662462296551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 4744762662462296551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 3252733973504856551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 3252733973504856551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 2989624276951896551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 2989624276951896551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 3628844259615576551-242ac11c-0001-012
Retrieved 5208 rearrangements for repertoire: 3628844259615576551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 3438706057421656551-242ac11c-0001-012
Retrieved 9530 rearrangements for repertoire: 3438706057421656551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 2197374609531736551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 2197374609531736551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 1993707260355416551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 1993707260355416551-242ac11c-0001-012
Retrieving rearrangements for repertoire: 2541616238306136551-242ac11c-0001-012
Retrieved 6512 rearrangements for repertoire: 2541616238306136551-242ac11c-0001-012
```

(continues on next page)

(continued from previous page)

```
Retrieving rearrangements for repertoire: 2366080924918616551-242ac11c-0001-012
Retrieved 10000 rearrangements for repertoire: 2366080924918616551-242ac11c-0001-012
```

The second python script loads the data from the files and generates a CDR3 amino acid length distribution that is grouped by the T cell subset. This study performs flow sorting to generate four T cell subsets: naive CD4+, naive CD8+, memory CD4+, memory CD8+. The script uses the repertoire metadata to determine the T cell subset for the rearrangement, tabulates the counts, normalizes them, and generates a grouped bar chart with the results:

```
# Run the analysis
python3 /airr-standards/docs/examples/api/analyze_data.py
```

The figure is placed in the `plot.png` file and should look like this:

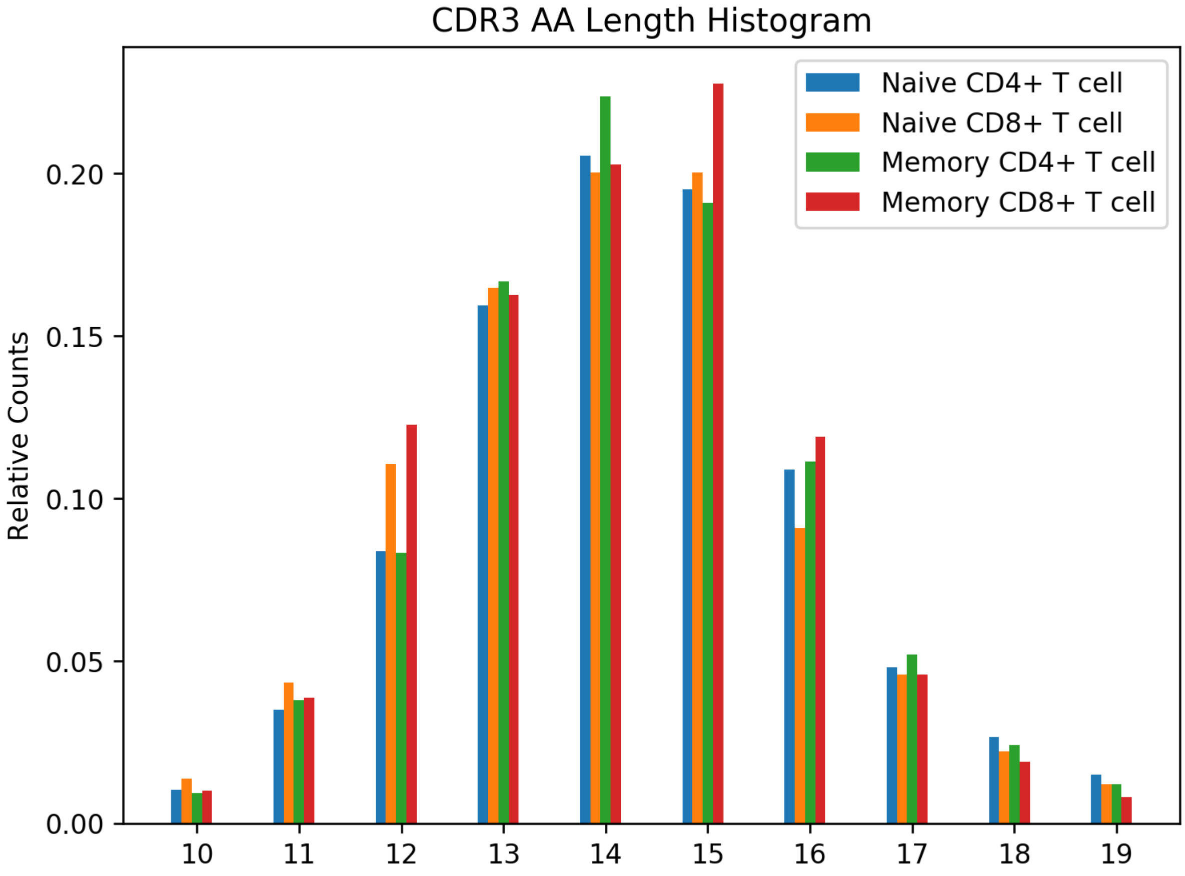


Fig. 3: **CDR3 AA Length Histogram grouped by T cell subsets.**

Scientific Query Scenarios for AIRR Data Commons API

The AIRR Common Repository Working Group (CRWG) has defined a number of sample scientific query scenarios to guide the design of the ADC API. The Design Decisions document lists the major design choices for the API, and the *API* is currently defined using the OpenAPI V2.0 Specification. This document describes the query examples with associated JSON definitions that can be submitted to an AIRR repository.

There are two main query endpoints in the API: /repertoire for querying MiAIRR-compliant study metadata and /rearrangement for querying rearrangement annotations. Most scientific queries will involve both endpoints. The basic workflow involves first querying /repertoire to get the list of repertoires that meet the search criteria on study, subject, and sample metadata. Secondly, the identifiers from the repertoires in the first query are passed to the /rearrangement endpoint along with any search criteria on the rearrangement annotations. The resultant rearrangements can be downloaded as JSON or in the *AIRR TSV format*.

Query Example 1

What human full length TCR-beta sequences have junction amino acid sequence: "CASSYIKLN"?

- The `JSON query definition` for /repertoire endpoint. The ontology identifier `9606` requests human and `TRB` is the locus of interest.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
                "content": {
                    "field":"subject.organism.id",
                    "value":"9606"
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"sample.pcr_target.pcr_target_locus",
                    "value":"TRB"
                }
            }
        ]
    }
}
```

- That query does not request full length sequences. We can enhance the `query` by adding a clause for the `sample.complete_sequences` field.

```
{
    "filters":{
        "op":"and",
        "content": [{
            "op":"=",
            "content": {
                "field":"subject.organism.id",
                "value":"9606"
            }
        },
        {
            "op":"=",
            "content": {
                "field":"sample.pcr_target.pcr_target_locus",
                "value":"TRB"
            }
        },
        {
            "op":"or",
```

(continues on next page)

(continued from previous page)

```
            "content":[{
                "op":"=",
                "content": {
                    "field":"sample.complete_sequences",
                    "value":"complete"
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"sample.complete_sequences",
                    "value":"complete+untemplated"
                }
            }]
        }]
    }
}
```

- The `JSON query definition` for /rearrangement endpoint. The repertoire identifiers (`repertoire_id`) in the query are just examples, you would replace them with the actual identifiers returned from the above repertoire query. The query performs an exact match of the junction amino acid sequence.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":[
                        "2366080924918616551-242ac11c-0001-012",
                        "2541616238306136551-242ac11c-0001-012",
                        "1993707260355416551-242ac11c-0001-012",
                        "1841923116114776551-242ac11c-0001-012"
                    ]
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
            }
        ]
    },
    "fields":["repertoire_id","sequence_id","v_call","productive"],
    "format":"tsv"
}
```

Query Example 2

What human full length IgH sequences have been found in patients with an autoimmune diagnosis.

- TO BE WRITTEN

Query Example 3

What is the antibody IG heavy chain V usage in people who have diabetes?

- TO BE WRITTEN

Query Example 4

Give me all the anti-HIV antibody sequences that use IGHV1-69 in HIV infected individuals?

- TO BE WRITTEN

Query Example 5

Repertoires from cancer patients where we have pre- and post-immunotherapy peripheral blood (or tumor biopsy).

- TO BE WRITTEN

Query Example 6

Return TCRs that score highly on a position weight matrix from subjects with a particular HLA allele that have been infected with TB.

- TO BE WRITTEN

Query Example 7

Repertoires from female patients with cancer.

- TO BE WRITTEN

2.2 Release Notes

2.2.1 Schema Release Notes

Version 1.3.0: May 28, 2020

Version 1.3 schema release.

New Schema:

1. Introduced the `Repertoire` Schema for describing study meta data.
2. Introduced the PCRTarget Schema for describing primer target locations.
3. Introduced the SampleProcessing Schema for describing experimental processing steps for a sample.
4. Replaced the SoftwareProcessing schema with the DataProcessing schema.

5. Introduced experimental schema for clonal clusters, lineage trees, tree nodes, and cells as Clone, Tree, Node, and Cell objects, respectively.

General Updates:

1. Added multiple additional attributes to a large number of schema propertes as AIRR extension attributes in the `x-airr` field. The new `Attributes` object contains definitions for these `x-airr` field attributes.
2. Added the top level `required` property to all relevant schema objects.
3. Added the `title` attribute containing the short, descriptive name to all relevant schema object fields.
4. Added an `example` attribute containing an example data value to multiple schema object fields.

AIRR Data Commons API:

1. Added OpenAPI V2 specification (`specs/adc-api.yaml`) for AIRR Data Commons API major version 1.

Ontology Support:

1. Added `Ontology` and `CURIEResolution` objects to support ontologies.
2. Added vocabularies/ontologies as JSON string for: Cell subset, Target substrate, Library generation method, Complete sequences, Physical linkage of different loci.

Rearrangement Schema:

1. Added the `complete_vdj` field to annotate whether a V(D)J alignment was full length.
2. Added the `junction_length_aa` field defining the length of the junction amino acid sequence.
3. Added the `repertoire_id`, `sample_processing_id`, and `data_processing_id` fields to serve as linkers to the appropriate metadata objects.
4. Added a controlled vocabulary to the `locus` field: `IGH`, `IGI`, `IGK`, `IGL`, `TRA`, `TRB`, `TRD`, `TRG`.
5. Deprecated the `rearrangement_set_id` and `germline_database` fields.
6. Deprecated `rearrangement_id` field and made the `sequence_id` field be the primary unique identifer for a rearrangement record, both in files and data repositories.
7. Added support secondary D gene rearrangement through the additional fields: `d2_call`, `d2_score`, `d2_identity`, `d2_support`, `d2_cigar np3`, `np3_aa`, `np3_length`, `n3_length`, `p5d2_length`, `p3d2_length`, `d2_sequence_start`, `d2_sequence_end`, `d2_germline_start`, `d2_germline_start`, `d2_alignment_start`, `d2_alignment_end`, `d2_sequence_alignment`, `d2_sequence_alignment_aa`, `d2_germline_alignment`, `d2_germline_alignment_aa`.
8. Updated field definitions with more concise V(D)J call descriptions.

Alignment Schema:

1. Deprecated the `rearrangement_set_id` and `germline_database` fields.
2. Added the `data_processing_id` field.

Study Schema:

1. Added the `study_type` field containing an ontology defined term for the study design.

Subject Schema:

1. Deprecated the `organism` field in favor of the new `species` field.
2. Deprecated the `age` field.
3. Introduced age ranges: `age_min`, `age_max`, and `age_unit`.

Diagnosis Schema:

1. Changed the type of the `disease_diagnosis` field from `string` to `Ontology`.

Sample Schema:

1. Changed the type of the `tissue` field from `string` to `Ontology`.

CellProcessing Schema:

1. Changed the type of the `cell_subset` field from `string` to `Ontology`.
2. Introduced the `cell_species` field which denotes the species from which the analyzed cells originate.

NucleicAcidProcessing Schema:

1. Defined the `template_class` field as type `string`.
2. Added a controlled vocabulary the `library_generation_method` field.
3. Changed the controlled vocabulary terms of `complete_sequences`. Replacing `complete & untemplated` with `complete+untemplated` and adding `mixed`.
4. Added the `pcr_target` field referencing the new `PCRTarget` schema object.

SequencingRun Schema:

1. Added the `sequencing_run_id` field which serves as the object identifer field.
2. Added the `sequencing_files` field which links to the RawSequenceData schema objects defining the raw read data.

RawSequenceData Schema:

1. Added the `file_type` field defining the sequence file type. This field is a controlled vocabulary restricted to: `fasta`, `fastq`.
2. Added the `paired_read_length` field defining mate-pair read lengths.
3. Defined the `read_direction` and `paired_read_direction` fields as type `string`.

DataProcessing Schema:

1. Replaces the SoftwareProcessing object.
2. Added `data_processing_id`, `primary_annotation`, `data_processing_files`, `germline_database` and `analysis_provenance_id` fields.

Version 1.2.1: Oct 5, 2018

Minor patch release.

1. Schema gene vs segment terminology corrections
2. Added `Info` object
3. Updated `cell_subset` URL in AIRR schema

Version 1.2.0: Aug 18, 2018

Peer reviewed released of the Rearrangement schema.

1. Definition change for the coordinate fields of the Rearrangement and Alignment schema. Coordinates are now defined as 1-based closed intervals, instead of 0-based half-open intervals (as previously defined in v1.1 of the schema).

2. Removed foreign study_id fields
3. Introduced keywords_study field

Version 1.1.0: May 3, 2018

Initial public released of the Rearrangement and Alignment schemas.

1. Added required and nullable constrains to AIRR schema.
2. Schema definitions for MiAIRR attributes and ontology.
3. Introduction of an x-airr object indicating if field is required by MiAIRR.
4. Rename rearrangement_set_id to data_processing_id.
5. Rename study_description to study_type.
6. Added physical_quantity format.
7. Raw sequencing files into separate schema object.
8. Rename Attributes object.
9. Added primary_annotation and repertoire_id.
10. Added diagnosis to repertoire object.
11. Added ontology for organism.
12. Added more detailed specification of sequencing_run, repertoire and rearrangement.
13. Added repertoire schema.
14. Rename definitions.yaml to airr-schema.yaml.
15. Removed c_call, c_score and c_cigar from required as this is not typical reference aligner output.
16. Renamed vdj_score, vdj_identity, vdj_evalue, and vdj_cigar to score, identity, evalue, and cigar.
17. Added missing c_identity and c_evalue fields to Rearrangement spec.
18. Swapped order of *N* and *S* operators in CIGAR string.
19. Some description clean up for consistency in Rearrangement spec.
20. Remove repeated objects in definitions.yaml.
21. Added Alignment object to definitions.yaml.
22. Updated MiARR format consistency check TSV with junction change.
23. Changed definition from functional to productive.

Version 1.0.1: Jan 9, 2018

MiAIRR v1 official release and initial draft of Rearrangement and Alignment schemas.

2.2.2 Python Library Release Notes

Version 1.3.0: May 30, 2020

1. Updated schema set to v1.3.
2. Added `load_repertoire`, `write_repertoire`, and `validate_repertoire` to `airr.interface` to read, write and validate Repertoire metadata, respectively.
3. Added `repertoire_template` to `airr.interface` which will return a complete repertoire object where all fields have `null` values.
4. Added `validate_object` to `airr.schema` that will validate a single repertoire object against the schema.
5. Extended the `airr-tools` commandline program to validate both rearrangement and repertoire files.

Version 1.2.1: October 5, 2018

1. Fixed a bug in the python reference library causing start coordinate values to be empty in some cases when writing data.

Version 1.2.0: August 17, 2018

1. Updated schema set to v1.2.
2. Several improvements to the `validate_rearrangement` function.
3. Changed behavior of all *airr.interface* functions to accept a file path (string) to a single Rearrangement TSV, instead of requiring a file handle as input.
4. Added `base` argument to `RearrangementReader` and `RearrangementWriter` to support optional conversion of 1-based closed intervals in the TSV to python-style 0-based half-open intervals. Defaults to conversion.
5. Added the custom exception `ValidationError` for handling validation checks.
6. Added the `validate` argument to `RearrangementReader` which will raise a `ValidationError` exception when reading files with missing required fields or invalid values for known field types.
7. Added `validate` argument to all type conversion methods in `Schema`, which will now raise a `ValidationError` exception for value that cannot be converted when set to `True`. When set `False` (default), the previous behavior of assigning `None` as the converted value is retained.
8. Added `validate_header` and `validate_row` methods to `Schema` and removed validations methods from `RearrangementReader`.
9. Removed automatic closure of file handle upon reaching the iterator end in `RearrangementReader`.

Version 1.1.0: May 1, 2018

Initial release.

2.2.3 R Library Release Notes

Version 1.3.0: May 26, 2020

1. Updated schema set to v1.3.

2. Added info slot to Schema object containing general schema information.

Version 1.2.0: August 17, 2018

1. Updated schema set to v1.2.
2. Changed defaults to base="1" for read and write functions.
3. Updated example TSV file with coordinate changes, addition of germline_alignment data and simplification of sequence_id values.

Version 1.1.0: May 1, 2018

Initial release.

2.3 AIRR Standards

Information about all of the AIRR Community standards.

2.3.1 Introduction to MiAIRR

Summary

One of the primary initiatives of the Adaptive Immune Receptor Repertoire (AIRR) Community has been to develop a set of metadata standards for the submission of AIRR sequencing datasets. This work has been carried out by the AIRR Community Minimal Standards Working Group. In order to support reproducibility, standard quality control, and data deposition in a common repository, the AIRR Community has agreed to six high-level data sets that will guide the publication, curation and sharing of AIRR-Seq data and metadata: Study and subject, sample collection, sample processing and sequencing, raw sequences, processing of sequence data, and processed AIRR sequences. The detailed data elements within these sets are defined *here* (Download as TSV).

Topics

MiAIRR Data Elements

The AIRR Community has agreed to six high-level data sets that will guide the publication, curation and sharing of AIRR-Seq data and metadata: Study and subject, sample collection, sample processing and sequencing, raw sequences, processing of sequence data, and processed AIRR sequences.

Download as TSV.

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / study	Study ID study_id	string *free text*	important	Unique ID assigned by study registry	PRJNA001

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / study	Study title `study_title`	string *free text*	important	Descriptive study title	Effects of sun light exposure of the Treg repertoire
1 / study	Study type `study_type`	Ontology *Ontology: { top_node: { id: NCIT:C63536, value: Study}}*	important	Type of study design	id: NCIT:C15197, value: Case-Control Study
1 / study	Study inclusion/exclusion criteria `inclusion_exclusion_criteria`	string *free text*	important	List of criteria for inclusion/exclusion for the study	Include: Clinical P. falciparum infection; Exclude: Seropositive for HIV
1 / study	Grant funding agency `grants`	string *free text*	important	Funding agencies and grant numbers	NIH, award number R01GM987654
1 / study	Contact information (data collection) `collected_by`	string *free text*	important	Full contact information of the data collector, i.e. the person who is legally responsible for data collection and release. This should include an e-mail address.	Dr. P. Stibbons, p.stibbons@unseenu.edu
1 / study	Lab name `lab_name`	string *free text*	important	Department of data collector	Department for Planar Immunology
1 / study	Lab address `lab_address`	string *free text*	important	Institution and institutional address of data collector	School of Medicine, Unseen University, Ankh-Morpork, Disk World
1 / study	Contact information (data deposition) `submitted_by`	string *free text*	important	Full contact information of the data depositor, i.e. the person submitting the data to a repository. This is supposed to be a short-lived and technical role until the submission is relased.	Adrian Turnipseed, a.turnipseed@unseenu.edu
1 / study	Relevant publications `pub_ids`	string *free text*	important	Publications describing the rationale and/or outcome of the study	PMID:85642
1 / study	Keywords for study `keywords_study`	array of string *Controlled vocabulary: contains_ig, contains_tcr, contains_single_cell, contains_paired_chain*	important	Keywords describing properties of one or more data sets in a study	[‘contains_ig’, ‘contains_paired_chain’]

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / subject	Subject ID `subject_id`	string *free text*	important	Subject ID assigned by submitter, unique within study	SUB856413
1 / subject	Synthetic library `synthetic`	boolean *true* | *false*	essential	TRUE for libraries in which the diversity has been synthetically generated (e.g. phage display)	
1 / subject	Organism `species`	*Ontology* *Ontology: { top_node: { id: NCBITAXON:7776, value: Gnathostomata}}*	essential	Binomial designation of subject's species	id: NCBITAXON:9606, value: Homo sapiens
1 / subject	Sex `sex`	string *Controlled vocabulary: male, female, pooled, hermaphrodite, intersex, not collected, not applicable*	important	Biological sex of subject	female
1 / subject	Age minimum `age_min`	number *positive number*	important	Specific age or lower boundary of age range.	60
1 / subject	Age maximum `age_max`	number *positive number*	important	Upper boundary of age range or equal to age_min for specific age. This field should only be null if age_min is null.	80
1 / subject	Age unit `age_unit`	*Ontology* *Ontology: { top_node: { id: UO:0000003, value: time unit}}*	important	Unit of age range	id: UO:0000036, value: year
1 / subject	Age event `age_event`	string *free text*	important	Event in the study schedule to which *Age* refers. For NCBI BioSample this MUST be *sampling*. For other implementations submitters need to be aware that there is currently no mechanism to encode to potential delta between *Age event* and *Sample collection time*, hence the chosen events should be in temporal proximity.	enrollment

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / subject	Ancestry population `ancestry_population`	string *free text*	important	Broad geographic origin of ancestry (continent)	list of continents, mixed or unknown
1 / subject	Ethnicity `ethnicity`	string *free text*	important	Ethnic group of subject (defined as cultural/language-based membership)	English, Kurds, Manchu, Yakuts (and other fields from Wikipedia)
1 / subject	Race `race`	string *free text*	important	Racial group of subject (as defined by NIH)	White, American Indian or Alaska Native, Black, Asian, Native Hawaiian or Other Pacific Islander, Other
1 / subject	Strain name `strain_name`	string *free text*	important	Non-human designation of the strain or breed of animal used	C57BL/6J
1 / subject	Relation to other subjects `linked_subjects`	string *free text*	important	Subject ID to which *Relation type* refers	SUB1355648
1 / subject	Relation type `link_type`	string *free text*	important	Relation between subject and *linked_subjects*, can be genetic or environmental (e.g.exposure)	father, daughter, household
1 / diagnosis and intervention	Study group description `study_group_description`	string *free text*	important	Designation of study arm to which the subject is assigned to	control
1 / diagnosis and intervention	Diagnosis `disease_diagnosis`	Ontology *Ontology: { top_node: { id: DOID:4, value: disease}}*	important	Diagnosis of subject	id: DOID:9538, value: multiple myeloma

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / diagnosis and intervention	Length of disease `disease_length`	string *free text*	important	Time duration between initial diagnosis and current intervention	23 months
1 / diagnosis and intervention	Disease stage `disease_stage`	string *free text*	important	Stage of disease at current intervention	Stage II
1 / diagnosis and intervention	Prior therapies for primary disease under study `prior_therapies`	string *free text*	important	List of all relevant previous therapies applied to subject for treatment of *Diagnosis*	melphalan/prednisone
1 / diagnosis and intervention	Immunogen/agent `immunogen`	string *free text*	important	Antigen, vaccine or drug applied to subject at this intervention	bortezomib

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / diagnosis and intervention	Intervention definition intervention	string *free text*	important	Description of intervention	systemic chemotherapy, 6 cycles, 1.25 mg/m2
1 / diagnosis and intervention	Other relevant medical history medical_history	string *free text*	important	Medical history of subject that is relevant to assess the course of disease and/or treatment	MGUS, first diagnosed 5 years prior
2 / sample	Biological sample ID sample_id	string *free text*	important	Sample ID assigned by submitter, unique within study	SUP52415
2 / sample	Sample type sample_type	string *free text*	important	The way the sample was obtained, e.g. fine-needle aspirate, organ harvest, peripheral venous puncture	Biopsy
2 / sample	Tissue tissue	*Ontology* *Ontology: { top_node: { id: UBERON:0010000, value: multicellular anatomical structure}}*	important	The actual tissue sampled, e.g. lymph node, liver, peripheral blood	id: UBERON:0002371, value: bone marrow
2 / sample	Anatomic site anatomic_site	string *free text*	important	The anatomic location of the tissue, e.g. Inguinal, femur	Iliac crest
2 / sample	Disease state of sample disease_state_sample	string *free text*	important	Histopathologic evaluation of the sample	Tumor infiltration
2 / sample	Sample collection time collection_time_point_relative	string *free text*	important	Time point at which sample was taken, relative to *Collection time event*	14 d
2 / sample	Collection time event collection_time_point_reference	string *free text*	important	Event in the study schedule to which *Sample collection time* relates to	Primary vaccination

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
2 / sample	Biomaterial provider `biomaterial_provider`	string *free text*	important	Name and address of the entity providing the sample	Tissues-R-Us, Tampa, FL, USA
3 / process (cell)	Tissue processing `tissue_processing`	string *free text*	important	Enzymatic digestion and/or physical methods used to isolate cells from sample	Collagenase A/Dnase I digested, followed by Percoll gradient
3 / process (cell)	Cell subset `cell_subset`	*Ontology* *Ontology: { top_node: { id: CL:0000542, value: lymphocyte}}*	important	Commonly-used designation of isolated cell population	id: CL:0000972, value: class switched memory B cell
3 / process (cell)	Cell subset phenotype `cell_phenotype`	string *free text*	important	List of cellular markers and their expression levels used to isolate the cell population	CD19+ CD38+ CD27+ IgM- IgD-
3 / process (cell)	Cell species `cell_species`	*Ontology* *Ontology: { top_node: { id: NCBITAXON:7776, value: Gnathostomata}}*	defined	Binomial designation of the species from which the analyzed cells originate. Typically, this value should be identical to *species*, if which case it SHOULD NOT be set explicitly. Howver, there are valid experimental setups in which the two might differ, e.g. chimeric animal models. If set, this key will overwrite the *species* information for all lower layers of the schema.	id: NCBITAXON:9606, value: Homo sapiens
3 / process (cell)	Single-cell sort `single_cell`	boolean *true* | *false*	important	TRUE if single cells were isolated into separate compartments	
3 / process (cell)	Number of cells in experiment `cell_number`	integer *positive integer*	important	Total number of cells that went into the experiment	1000000
3 / process (cell)	Number of cells per sequencing reaction `cells_per_reaction`	integer *positive integer*	important	Number of cells for each biological replicate	50000
3 / process (cell)	Cell storage `cell_storage`	boolean *true* | *false*	important	TRUE if cells were cryo-preserved between isolation and further processing	True

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (cell)	Cell quality `cell_quality`	string *free text*	important	Relative amount of viable cells after preparation and (if applicable) thawing	90% viability as determined by 7-AAD
3 / process (cell)	Cell isolation / enrichment procedure `cell_isolation`	string *free text*	important	Description of the procedure used for marker-based isolation or enrich cells	Cells were stained with fluorochrome labeled antibodies and then sorted on a FlowMerlin (CE) cytometer.
3 / process (cell)	Processing protocol `cell_processing_protocol`	string *free text*	important	Description of the methods applied to the sample including cell preparation/ isolation/enrichment and nucleic acid extraction. This should closely mirror the Materials and methods section in the manuscript.	Stimulated wih anti-CD3/anti-CD28
3 / process (nucleic acid)	Target substrate `template_class`	string *Controlled vocabulary: DNA, RNA*	essential	The class of nucleic acid that was used as primary starting material for the following procedures	RNA
3 / process (nucleic acid)	Target substrate quality `template_quality`	string *free text*	important	Description and results of the quality control performed on the template material	RIN 9.2
3 / process (nucleic acid)	Template amount `template_amount`	string *free text*	important	Amount of template that went into the process	1000 ng

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (nucleic acid)	Library generation method `library_generation_method`	string *Controlled vocabulary: PCR, RT(RHP)+PCR, RT(oligo-dT)+PCR, RT(oligo-dT)+TS+PCR, RT(oligo-dT)+TS(UMI)+PCR, RT(specific)+PCR, RT(specific)+TS+PCR, RT(specific)+TS(UMI)+PCR, RT(specific+UMI)+PCR, RT(specific+UMI)+TS+PCR, RT(specific)+TS, other*	essential	Generic type of library generation	RT(oligo-dT)+TS(UMI)+PCR
3 / process (nucleic acid)	Library generation protocol `library_generation_protocol`	string *free text*	important	Description of processes applied to substrate to obtain a library that is ready for sequencing	cDNA was generated using
3 / process (nucleic acid)	Protocol IDs `library_generation_kit_version`	string *free text*	important	When using a library generation protocol from a commercial provider, provide the protocol version number	v2.1 (2016-09-15)

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (nucleic acid)	Complete sequences `complete_sequences`	string *Controlled vocabulary: partial, complete, complete+untemplated, mixed*	essential	To be considered *complete*, the procedure used for library construction MUST generate sequences that 1) include the first V gene codon that encodes the mature polypeptide chain (i.e. after the leader sequence) and 2) include the last complete codon of the J gene (i.e. 1 bp 5' of the J->C splice site) and 3) provide sequence information for all positions between 1) and 2). To be considered *complete & untemplated*, the sections of the sequences defined in points 1) to 3) of the previous sentence MUST be untemplated, i.e. MUST NOT overlap with the primers used in library preparation. *mixed* should only be used if the procedure used for library construction will likely produce multiple categories of sequences in the given experiment. It SHOULD NOT be used as a replacement of a NULL value.	partial

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (nucleic acid)	Physical linkage of different rearrangements `physical_linkage`	string *Controlled vocabulary: none, hetero_head-head, hetero_tail-head, hetero_prelinked*	essential	In case an experimental setup is used that physically links nucleic acids derived from distinct *Rearrangements* before library preparation, this field describes the mode of that linkage. All *hetero_** terms indicate that in case of paired-read sequencing, the two reads should be expected to map to distinct IG/TR loci. **_head-head* refers to techniques that link the 5' ends of transcripts in a single-cell context. **_tail-head* refers to techniques that link the 3' end of one transcript to the 5' end of another one in a single-cell context. This term does not provide any information whether a continuous reading-frame between the two is generated. **_prelinked* refers to constructs in which the linkage was already present on the DNA level (e.g. scFv).	hetero_head-head
3 / process (nucleic acid [pcr])	Target locus for PCR `pcr_target_locus`	string *Controlled vocabulary: IGH, IGI, IGK, IGL, TRA, TRB, TRD, TRG*	important	Designation of the target locus. Note that this field uses a controlled vocubulary that is meant to provide a generic classification of the locus, not necessarily the correct designation according to a specific nomenclature.	IGK
3 / process (nucleic acid [pcr])	Forward PCR primer target location `forward_pcr_primer_target_location`	string *free text*	important	Position of the most distal nucleotide templated by the forward primer or primer mix	IGHV, +23
3 / process (nucleic acid [pcr])	Reverse PCR primer target location `reverse_pcr_primer_target_location`	string *free text*	important	Position of the most proximal nucleotide templated by the reverse primer or primer mix	IGHG, +57

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (sequencing)	Batch number `sequencing_run_id`	string *free text*	important	ID of sequencing run assigned by the sequencing facility	160101_M01234
3 / process (sequencing)	Total reads passing QC filter `total_reads_passing_qc_filter`	integer *positive integer*	important	Number of usable reads for analysis	10365118
3 / process (sequencing)	Sequencing platform `sequencing_platform`	string *free text*	important	Designation of sequencing instrument used	Alumina LoSeq 1000
3 / process (sequencing)	Sequencing facility `sequencing_facility`	string *free text*	important	Name and address of sequencing facility	Seqs-R-Us, Vancouver, BC, Canada
3 / process (sequencing)	Date of sequencing run `sequencing_run_date`	string *free text*	important	Date of sequencing run	2016-12-16
3 / process (sequencing)	Sequencing kit `sequencing_kit`	string *free text*	important	Name, manufacturer, order and lot numbers of sequencing kit	FullSeq 600, Alumina, #M123456C0, 789G1HK
4 / data (raw reads)	Raw sequencing data file type `file_type`	string *Controlled vocabulary: fasta, fastq*	important	File format for the raw reads or sequences	
4 / data (raw reads)	Raw sequencing data file name `filename`	string *free text*	important	File name for the raw reads or sequences. The first file in paired-read sequencing.	MS10R-NMonson-C7JR9_S1_R1_001.fastq

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
4 / data (raw reads)	Read direction read_direction	string *Controlled vocabulary: forward, reverse, mixed*	important	Read direction for the raw reads or sequences. The first file in paired-read sequencing.	forward
4 / process (sequencing)	Forward read length read_length	integer *positive integer*	important	Read length in bases for the first file in paired-read sequencing	300
4 / data (raw reads)	Paired raw sequencing data file name paired_filename	string *free text*	important	File name for the second file in paired-read sequencing	MS10R-NMonson-C7JR9_S1_R2_001.fastq
4 / data (raw reads)	Paired read direction paired_read_direction	string *Controlled vocabulary: forward, reverse, mixed*	important	Read direction for the second file in paired-read sequencing	reverse
4 / process (sequencing)	Paired read length paired_read_length	integer *positive integer*	important	Read length in bases for the second file in paired-read sequencing	300
5 / process (computational)	Software tools and version numbers software_versions	string *free text*	important	Version number and / or date, include company pipelines	IgBLAST 1.6
5 / process (computational)	Paired read assembly paired_reads_assembly	string *free text*	important	How paired end reads were assembled into a single receptor sequence	PandaSeq (minimal overlap 50, threshold 0.8)
5 / process (computational)	Quality thresholds quality_thresholds	string *free text*	important	How sequences were removed from (4) based on base quality scores	Average Phred score >=20

Continued on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
5 / process (computational)	Primer match cutoffs `primer_match_cutoffs`	string *free text*	important	How primers were identified in the sequences, were they removed/masked/etc?	Hamming distance <= 2
5 / process (computational)	Collapsing method `collapsing_method`	string *free text*	important	The method used for combining multiple sequences from (4) into a single sequence in (5)	MUSCLE 3.8.31
5 / process (computational)	Data processing protocols `data_processing_protocols`	string *free text*	important	General description of how QC is performed	Data was processed using [. . .]
5 / data (processed sequence)	V(D)J germline reference database `germline_database`	string *free text*	important	Source of germline V(D)J genes with version number or date accessed.	ENSEMBL, Homo sapiens build 90, 2017-10-01

MiAIRR-to-NCBI Implementation

Authors Christian E. Busse, Florian Rubelt and Syed Ahmad Chan Bukhari

Guide for submission of AIRR-seq data to NCBI

This site provides a detailed “how-to” guide for submission of AIRR-seq data to **NCBI repositories** (BioProject, BioSample, SRA and GenBank). For other implementations of the MiAIRR standard see here.

One of the primary initiatives of the AIRR (Adaptive Immune Receptor Repertoire) Community has been to develop a set of metadata standards for the submission of immune receptor repertoire sequencing datasets. This work has been carried out by the AIRR Community Standards Working Group. In order to support reproducibility, standard quality control, and data deposition in a common repository, the AIRR Community has agreed to six high-level data sets that will guide the publication, curation and sharing of AIRR-Seq data and metadata: Study and subject, sample collection, sample processing and sequencing, raw sequences, processing of sequence data, and processed AIRR sequences. The detailed data elements within these sets are defined *here* (`Download as TSV`). The association between these AIRR sets, the associated data elements, and each of the NCBI repositories is shown below:

Submission of AIRR sequencing data and metadata to NCBI’s public data repositories consists of five sequential steps:

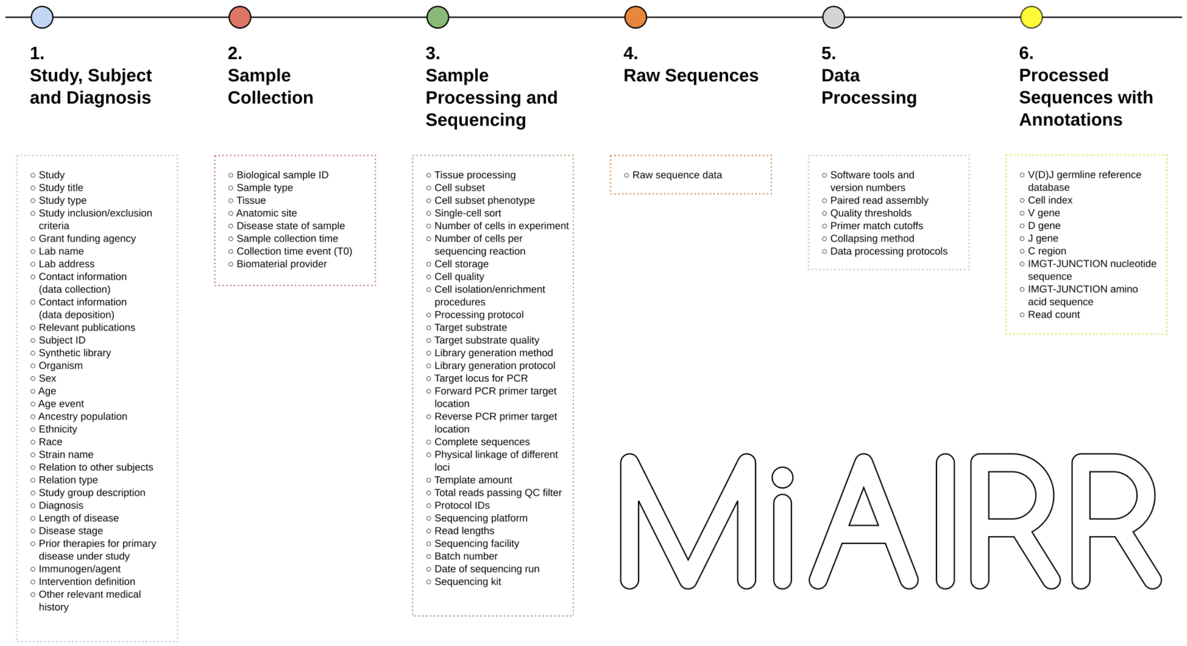
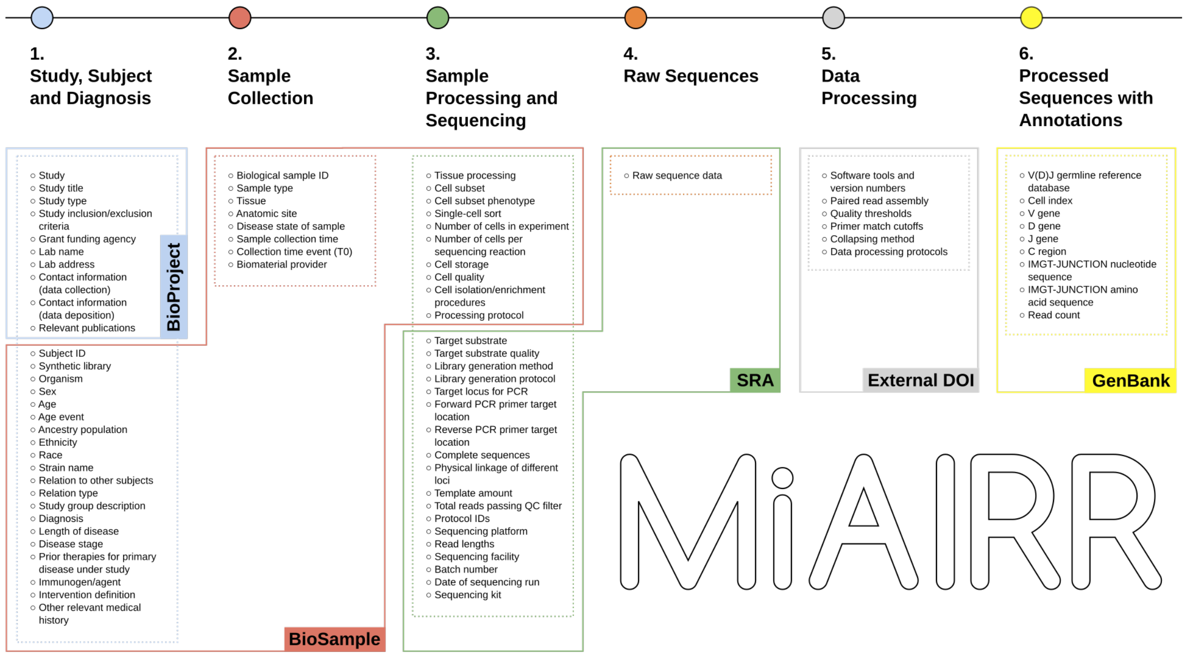


Fig. 4: Schema of MiAIRR data sets and the individual data elements of each set.



1. Submit study information to NCBI BioProject using the NCBI web interface.
2. Submit sample-level information to the NCBI BioSample repository using the AIRR-BioSample templates.
3. Submit raw sequencing data to NCBI SRA using the AIRR-SRA data templates.
4. Generate a DOI for the protocol describing how raw sequencing data were processed using Zenodo.
5. Submit processed sequencing data with sequence-level annotations to GenBank using AIRR feature tags.

The *submission manual* provides step-by-step instructions on carrying out these steps for an AIRR study submission.

MiAIRR-to-NCBI Submission Manual

Scope of this document

Provide a user manual describing the submission of AIRR data using the NCBI reference implementation described in [Rubelt_2017]. This implementation uses NCBI's BioProject, BioSample, Sequence Read Archive (SRA) and GenBank repositories and metadata standards to report AIRR data.

Step 1. MiAIRR data submission to BioProject, BioSample and SRA

Since we propose to include a combination of raw and processed sequence data, the AIRR standard will sometimes need to be distributed and linked across multiple repositories (e.g., data in SRA linked to related data in GenBank). Besides, the data elements that comprise the standard will be mapped to ontologies in BioPortal through NIH CDE (Common Data Element) terms. These linkages will support more sophisticated validation and logical inference.

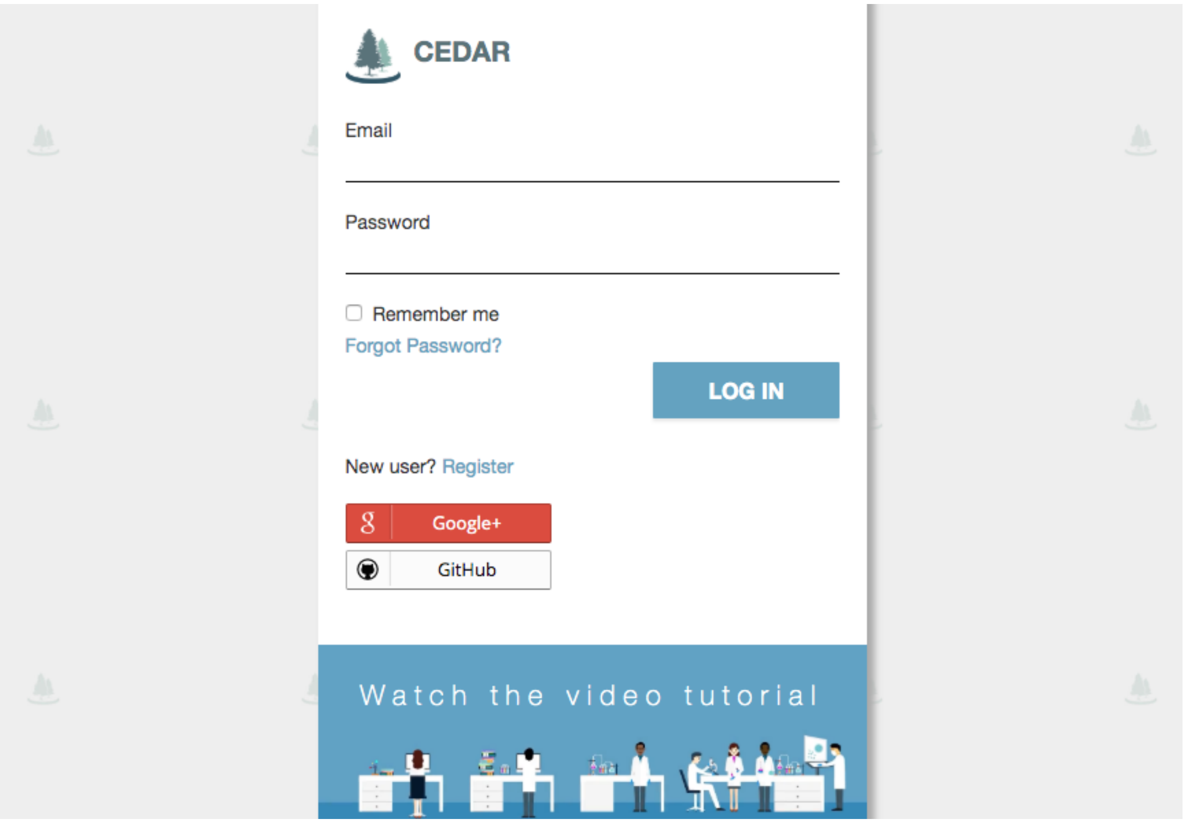
There are three main alternatives to submit raw AIRR data/metadata to NCBI repositories: (1) CEDAR's CAIRR pipeline; (2) NCBI's Web interface; and (3) NCBI's FTP server. These alternatives are described below:

Option 1. Submission via the CEDAR system (CAIRR submission pipeline)

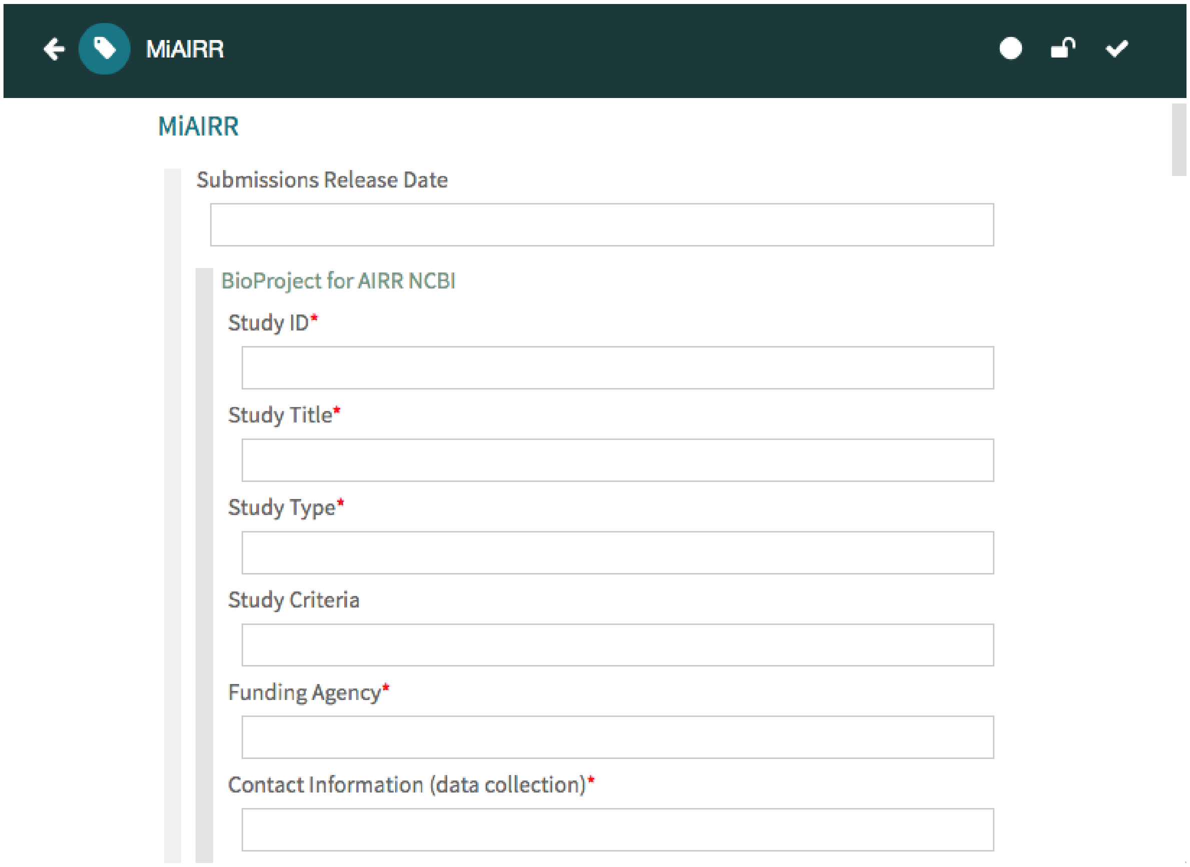
CEDAR's CAIRR submission pipeline helps investigators and curators to edit and validate ontology-controlled metadata. This pipeline provides a seamless interface to transmit SRA datasets to the NCBI SRA and BioSample repositories from the CEDAR Workbench. The pipeline can be directly be accessed at http://cairr.airr-community.org. Note that the CEDAR template and template elements used by this pipeline are publicly available in the following CEDAR folder: All/Shared/Shared by CEDAR/MiAIRR.

Submission steps:

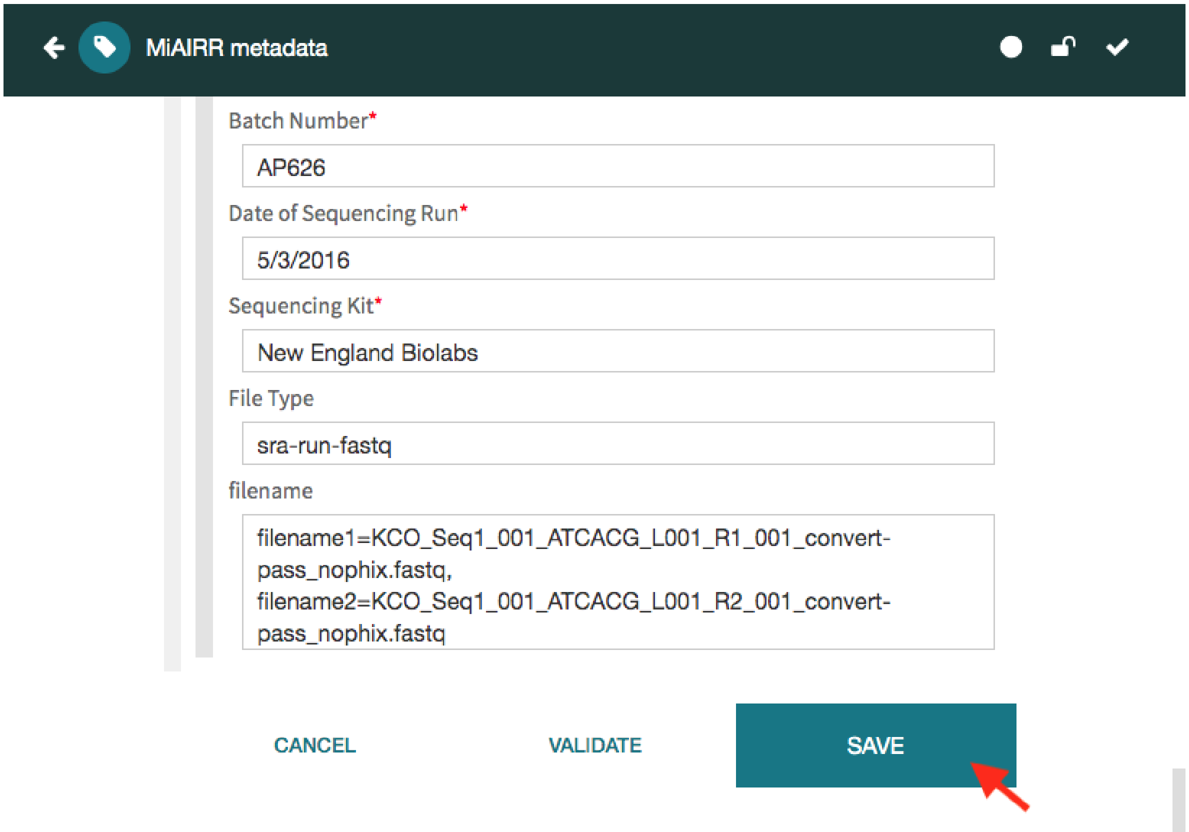
1. Open CEDAR's MiAIRR template by clicking on http://cairr.airr-community.org. If you are not already logged in, this will take you to the CEDAR login panel. If you are a new user, you will have to create an account on the CEDAR Workbench by clicking here.



2. After logging in into the system, you will see the 'MiAIRR' template. Fill out the template fields with your metadata. Fields with an asterisk (*) are mandatory. Your submission will fail if any mandatory fields are not completed. If information is unavailable for any mandatory field, please enter 'not collected', 'not applicable' or 'missing' as appropriate. Note that you will need to enter a BioProject ID into the field 'Study ID'. If you do not have a BioProject yet, you can create one at *https://submit.ncbi.nlm.nih.gov/subs/bioproject/*



3. Once your metadata is complete, click on the ‘Save’ button to save your metadata into your workspace. You will see a message in a green box confirming that your metadata have been successfully saved, as well as a message in a yellow box letting you know that your metadata have been saved to your personal workspace.


MiAIRR metadata

Batch Number*

AP626

Date of Sequencing Run*

5/3/2016

Sequencing Kit*

New England Biolabs

File Type

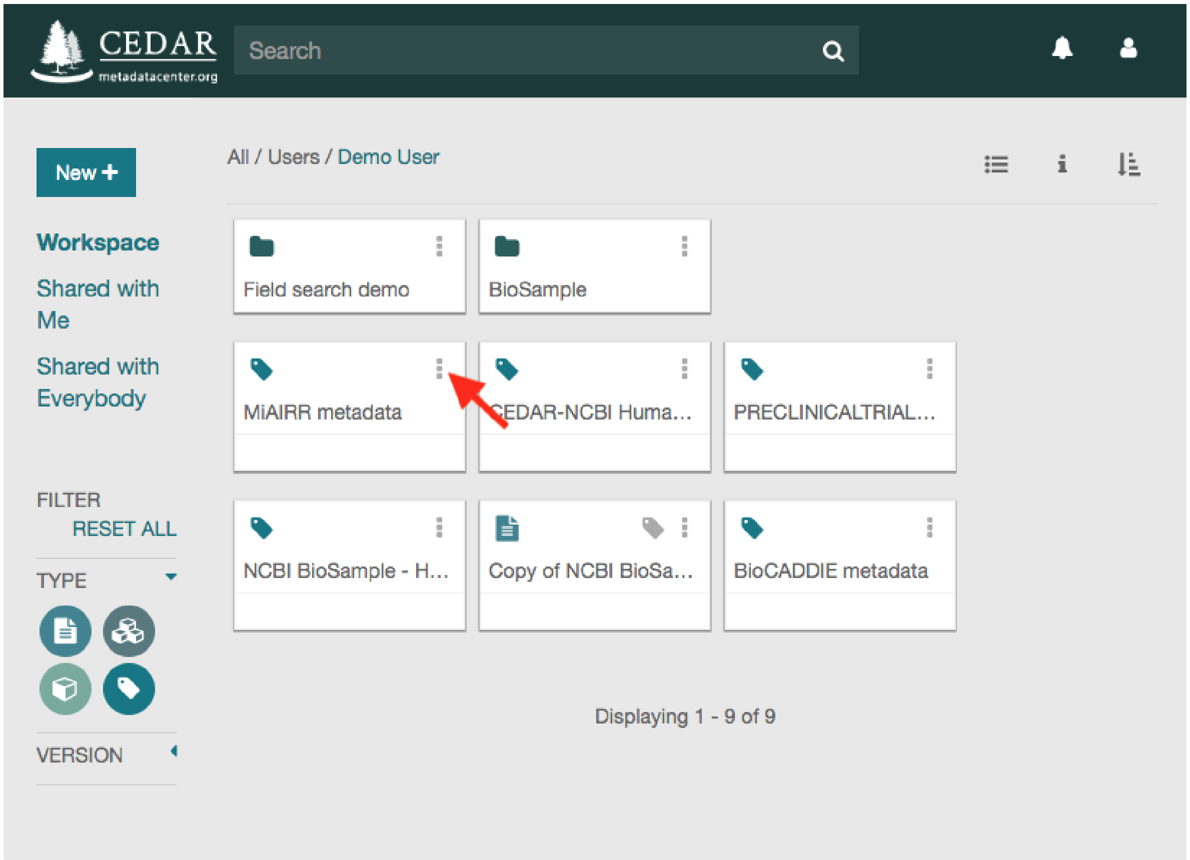
sra-run-fastq

filename

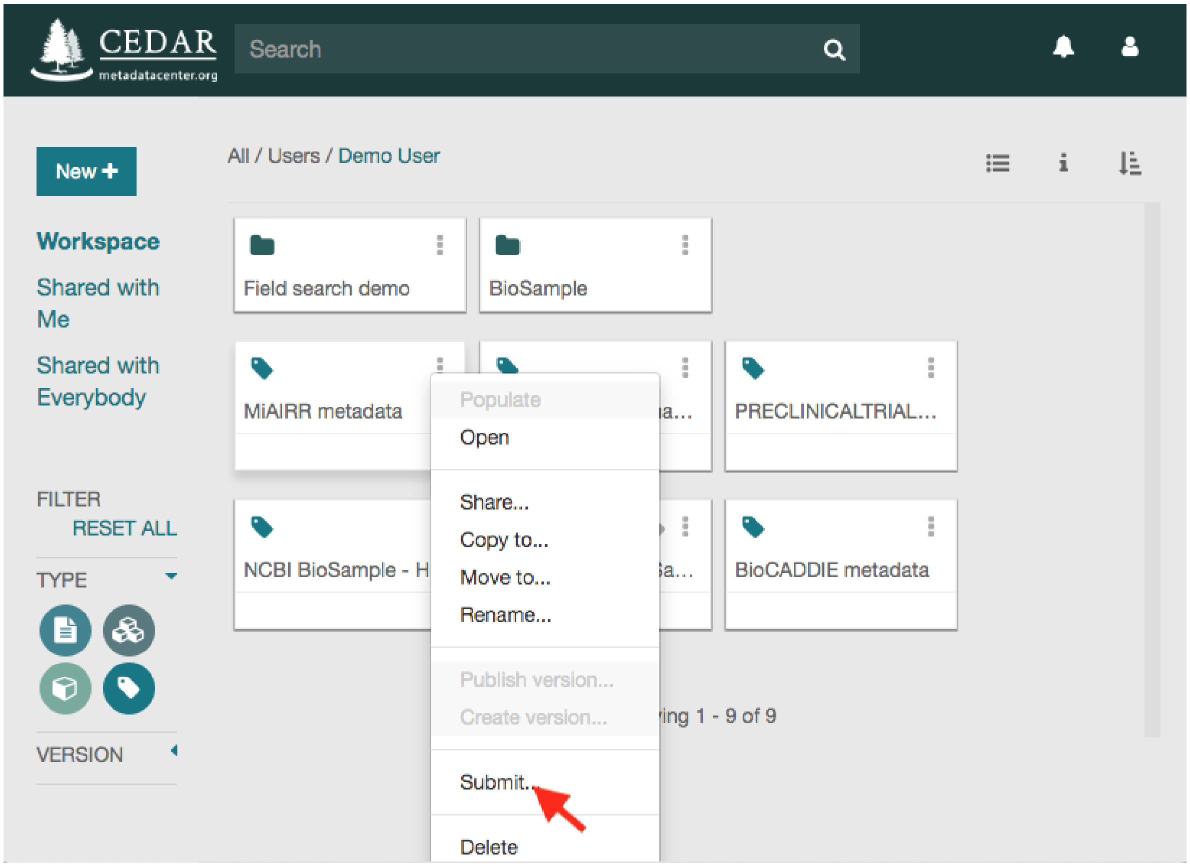
filename1=KCO_Seq1_001_ATCACG_L001_R1_001_convert-pass_nophix.fastq, filename2=KCO_Seq1_001_ATCACG_L001_R2_001_convert-pass_nophix.fastq

4. Go to your personal workspace by clicking on the left arrow (top left corner) and then on the 'Workspace' link, or by just clicking on: https://cedar.metadatacenter.org

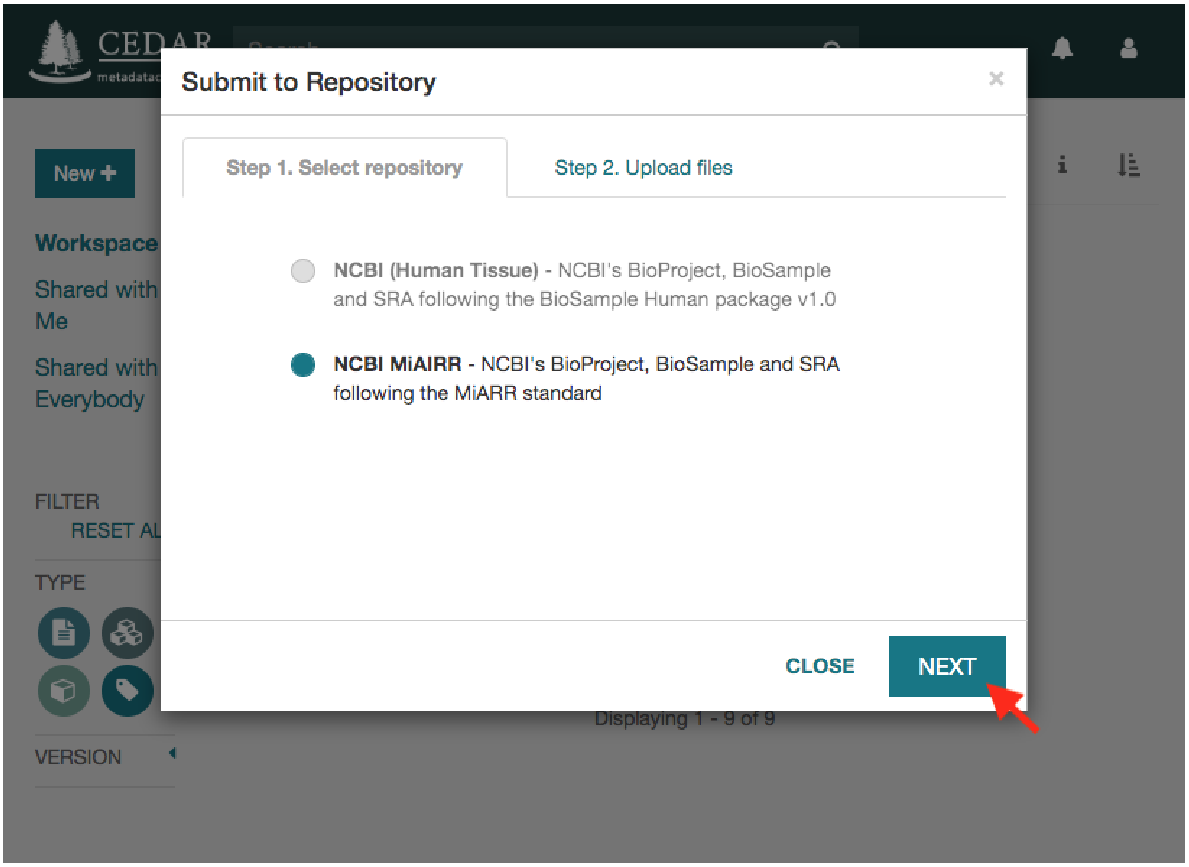
5. Once in your workspace, you will see a metadata file called 'MiAIRR metadata'. That file contains the metadata that you have just created and that you want to submit to the NCBI. Click on the three vertical dots on the top-right corner of the file icon to see the available file options.



6. Click on the ‘Submit’ option to open the submission dialog.



7. The ‘NCBI MiAIRR’ option will be automatically selected. Click on ‘Next’ to go to the next step.



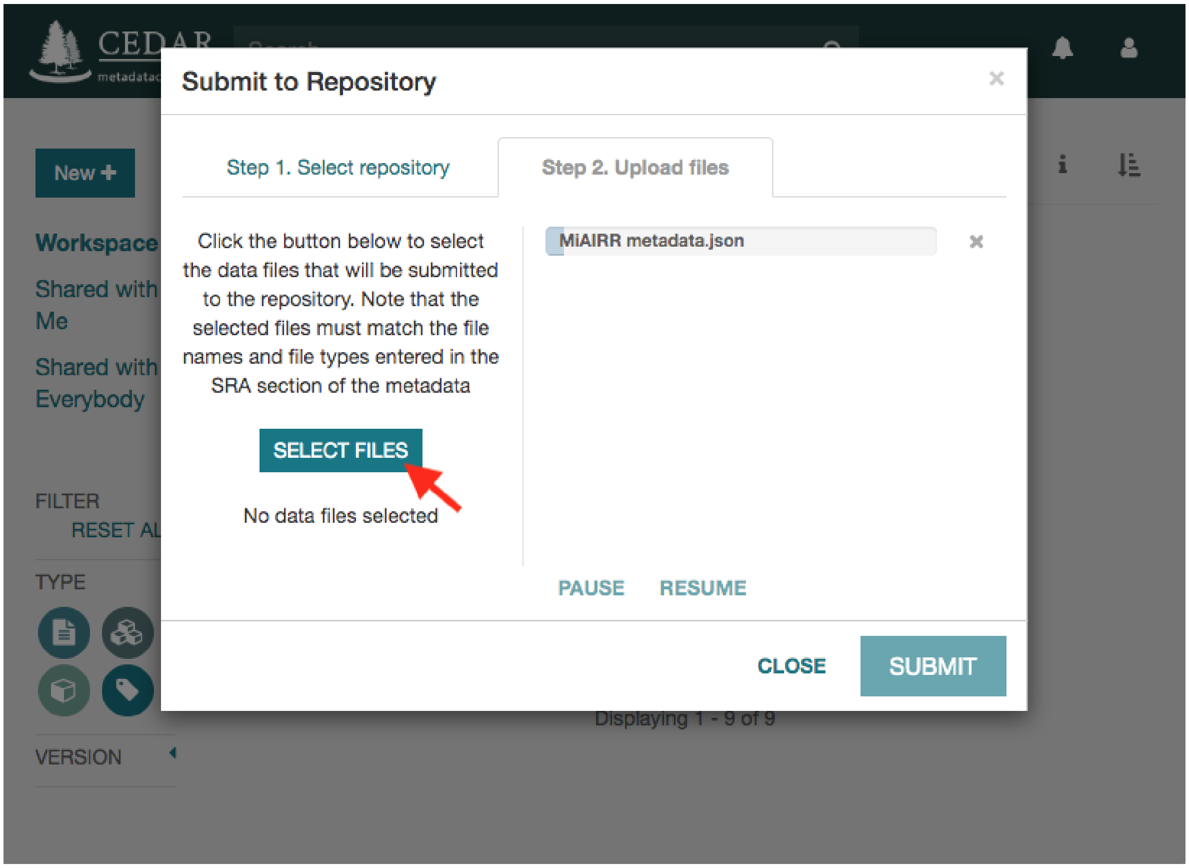
Submit to Repository

Step 1. Select repository | Step 2. Upload files

○ **NCBI (Human Tissue)** - NCBI's BioProject, BioSample and SRA following the BioSample Human package v1.0

● **NCBI MiAIRR** - NCBI's BioProject, BioSample and SRA following the MiARR standard

8. Click on the 'Select Files' button to upload the data files. Note that the names of the selected files must match the names in the metadata file. Otherwise, you will receive an error message when trying to start the submission.



Submit to Repository

Step 1. Select repository

Step 2. Upload files

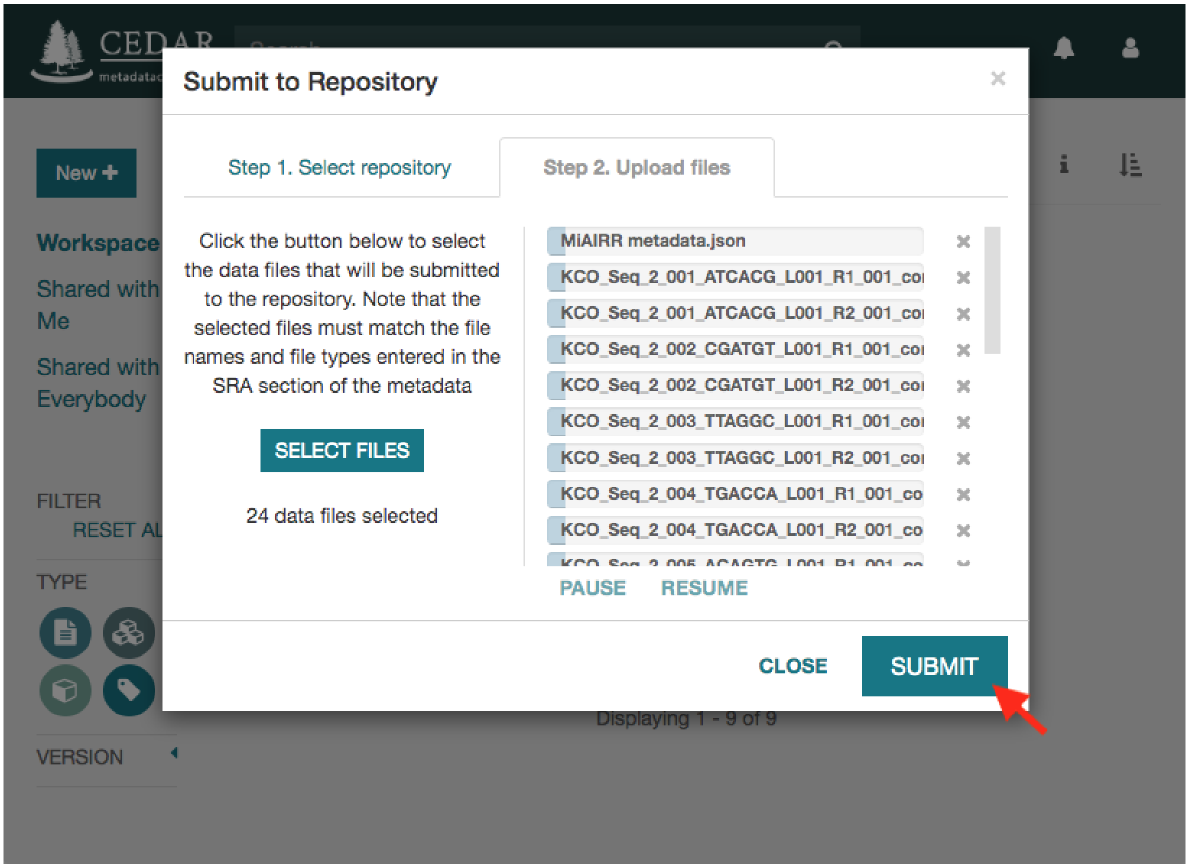
Click the button below to select the data files that will be submitted to the repository. Note that the selected files must match the file names and file types entered in the SRA section of the metadata

SELECT FILES

No data files selected

MiAIRR metadata.json

9. Click on the ‘Submit’ button to start the submission. If there are not validation errors, the selected data files and the corresponding metadata will be uploaded to the NCBI servers.



Submit to Repository

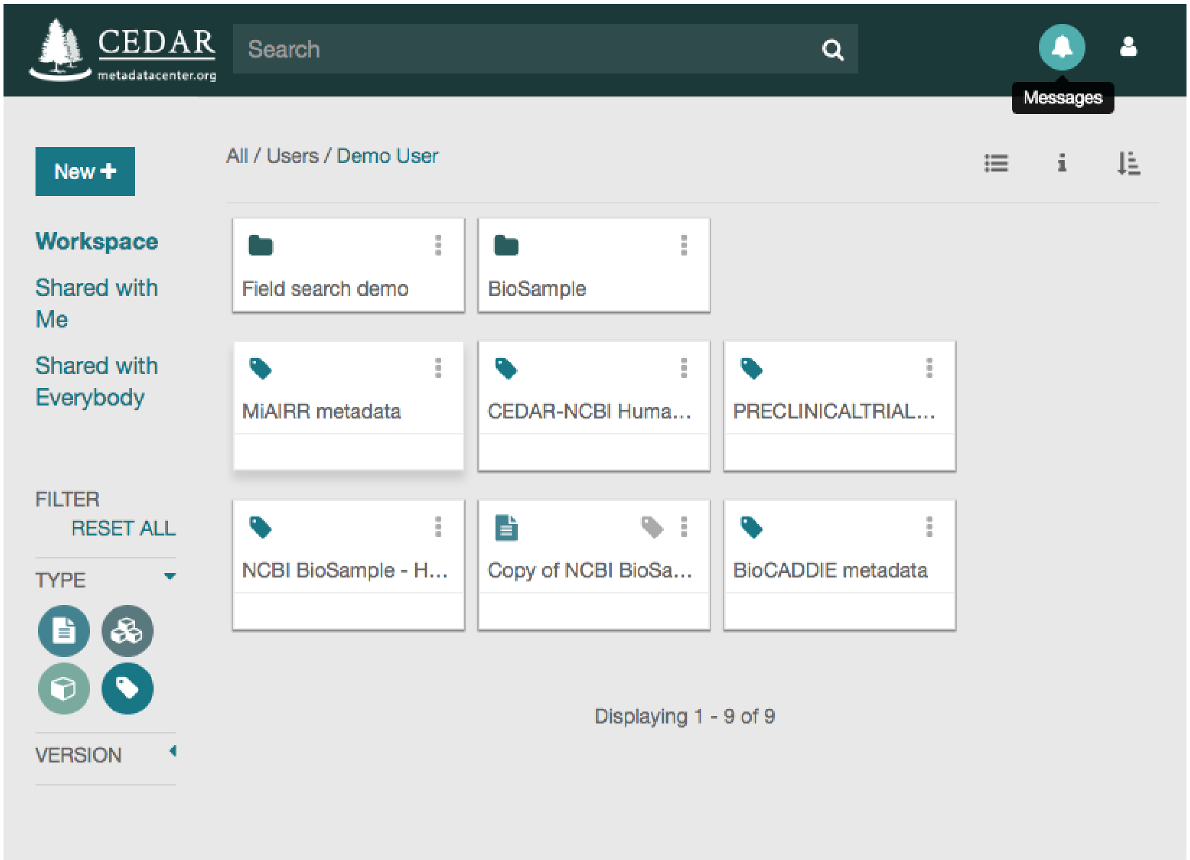
Click the button below to select the data files that will be submitted to the repository. Note that the selected files must match the file names and file types entered in the SRA section of the metadata

24 data files selected

- MiAIRR metadata.json
- KCO_Seq_2_001_ATCACG_L001_R1_001_co
- KCO_Seq_2_001_ATCACG_L001_R2_001_co
- KCO_Seq_2_002_CGATGT_L001_R1_001_co
- KCO_Seq_2_002_CGATGT_L001_R2_001_co
- KCO_Seq_2_003_TTAGGC_L001_R1_001_co
- KCO_Seq_2_003_TTAGGC_L001_R2_001_co
- KCO_Seq_2_004_TGACCA_L001_R1_001_co
- KCO_Seq_2_004_TGACCA_L001_R2_001_co
- KCO_Seq_2_005_ACAGTG_L001_R1_001_co

Displaying 1 - 9 of 9

10. Note that the submission may take several hours or even days to be processed by the NCBI. Meanwhile, you will receive status messages about your submission in your workspace (messages icon).



11. Proceed with deposit of processed data, below.

Citing the CAIRR pipeline

Bukhari, Syed Ahmad Chan, Martin J. O'Connor, Marcos Martínez-Romero, Attila L. Egyedi, Debra Debra Willrett, John Graybeal, Mark A. Musen, Florian Rubelt, Kei H. Cheung, and Steven H. Kleinstein. The CAIRR pipeline for submitting standards-compliant B and T cell receptor repertoire sequencing studies to the NCBI. Frontiers in Immunology 9 (2018): 1877. DOI: 10.3389/fimmu.2018.01877

Tell Us About It

Please let us know how it went! If you are willing, we would love to have your comments in a short survey, it should just take 5 minutes or so. We also welcome entry of issues and requests in our GitHub repository, and emails can be sent to cedar-users@lists.stanford.edu. Both of these resources are publicly visible.

Support or Contact

Having trouble with NCBI submission process through our pipeline? Please email to Syed Ahmad Chan Bukhari or to Marcos Martínez-Romero and we will help you sort it out.

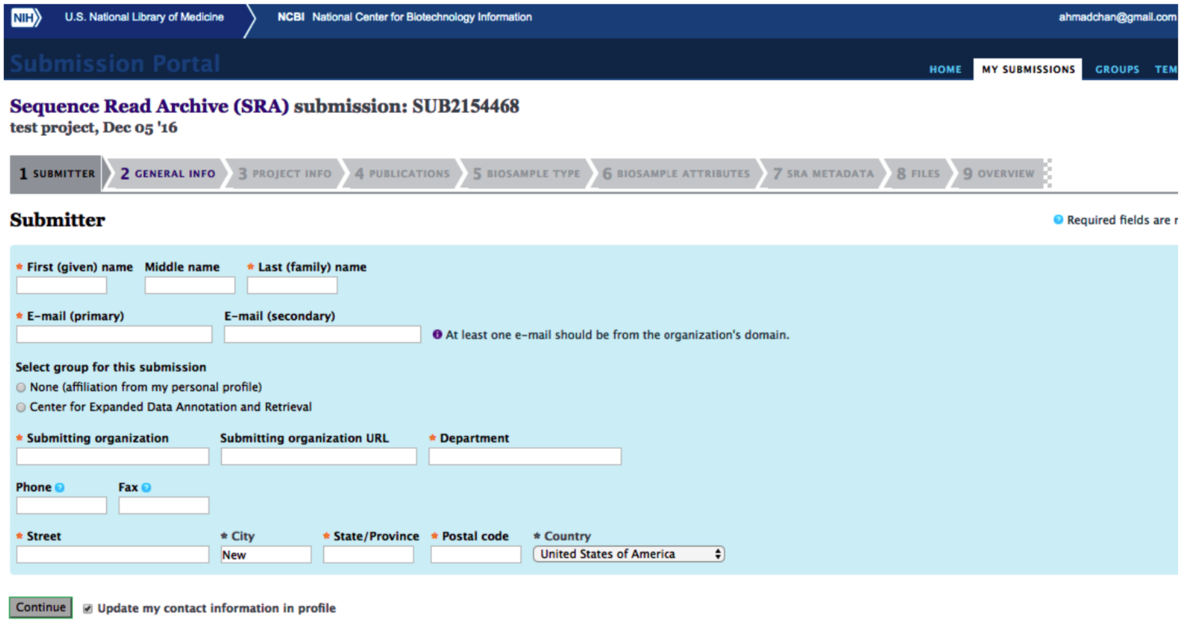
Option 2. Submission via NCBI's web interface

To facilitate AIRR data submissions to NCBI repositories, we have developed the NCBI-compliant metadata submission templates both for single and bulk AIRR data submissions. NCBI provides a web-based interface to create a BioProject and allows to BioSample, Sequence Read Archive (SRA) and GenBank metadata via tab-delimited files for single BioProject related data files submission.

Submitting AIRR data and associated metadata to the Bioproject, BioSample and SRA repositories via NCBI's web interface follows in general the submission procedure described in [NCBI_NBK47528], but uses AIRR-specific template for metadata submission:

1. Go to https://submit.ncbi.nlm.nih.gov/subs/sra/ and login with your NCBI account (create an account if necessary).

2. Click on "create new submission". You will see a form as below. Fill the form with required information and click on "continue".



Sequence Read Archive (SRA) submission: SUB2154468
test project, Dec 05 '16

Submitter

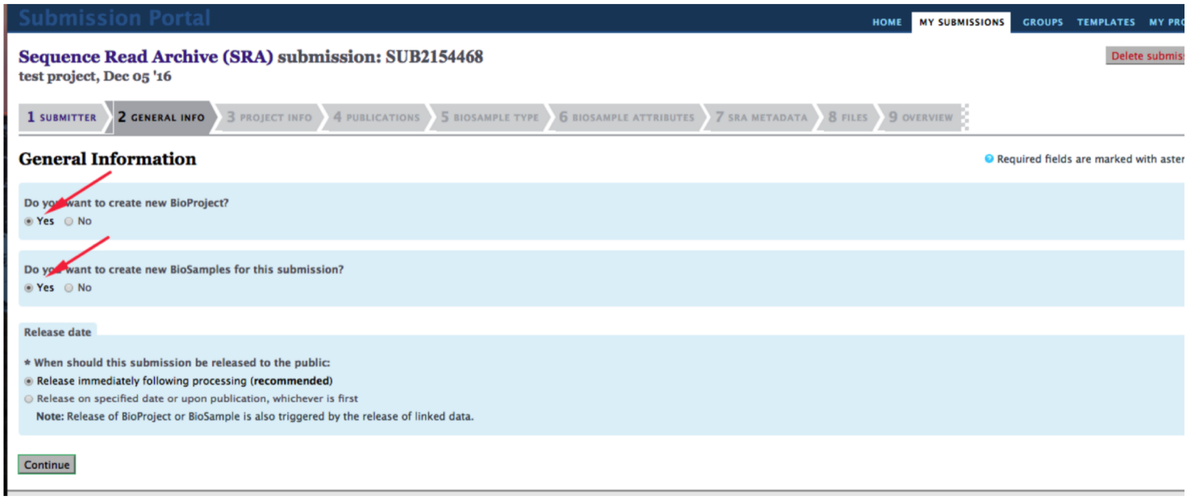
At least one e-mail should be from the organization's domain.

Select group for this submission
○ None (affiliation from my personal profile)
○ Center for Expanded Data Annotation and Retrieval

City: New
Country: United States of America

☑ Update my contact information in profile

3. If you are submitting for the first time, check "Yes" on the "new BioProject" or "new BioSample" options to create a new project or sample, respectively.



Sequence Read Archive (SRA) submission: SUB2154468
test project, Dec 05 '16

General Information

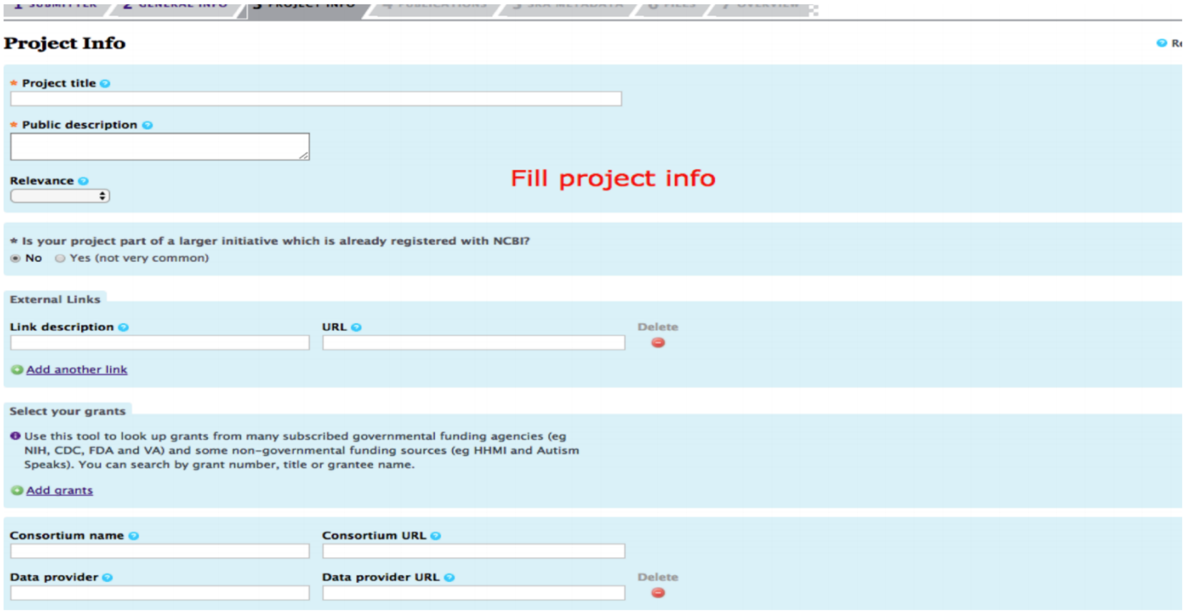
Do you want to create new BioProject?
● Yes ○ No

Do you want to create new BioSamples for this submission?
● Yes ○ No

Release date

* When should this submission be released to the public:
● Release immediately following processing (recommended)
○ Release on specified date or upon publication, whichever is first
Note: Release of BioProject or BioSample is also triggered by the release of linked data.

4. Fill in the project information. Add as much relevant information you can add in description. It will help later in searching the particular submission.



Project Info
Re
* Project title
* Public description
Relevance
Fill project info
* Is your project part of a larger initiative which is already registered with NCBI?
● No ○ Yes (not very common)
External Links
Link description
URL
Delete
Add another link
Select your grants
Use this tool to look up grants from many subscribed governmental funding agencies (eg NIH, CDC, FDA and VA) and some non-governmental funding sources (eg HHMI and Autism Speaks). You can search by grant number, title or grantee name.
Add grants
Consortium name
Consortium URL
Data provider
Data provider URL
Delete

5. The AIRR BioSample template is not yet listed on the NCBI website. The template sheet `AIRR_BioSample_V1.0.xls` can be downloaded from https://github.com/airr-community/airr-standards/tree/master/NCBI_implementation/templates_XLS. Fill in the required field and save the file as *tab-delimited* text file (.TSV format), then upload it.

6. To submit the SRA metadata use the `AIRR_SRA_v1.0.xls` file. Make sure that the column `sample_name` uses sample names that match the record in the BioSample template (if new BioSamples are being submitted) or a previously entered record. Also this file must be saved as *tab-delimited* text file for upload.

7. Submit the raw sequence file.

8. Complete the submission.

9. Proceed with deposit of processed data, below.

Option 3. Submission via NCBI's FTP server, using a predefined XML template

In addition to the web interface, NCBI provides an FTP-based solution to submit bulk metadata. The corresponding AIRR XML templates can be found under https://github.com/airr-community/airr-standards/tree/master/NCBI_implementation/templates_XLS. Otherwise users should refer to the current SRA file upload manual https://www.ncbi.nlm.nih.gov/sra/docs/submitfiles/. Users planning to frequently submit AIRR-seq data to SRA using scripts to generate the XML files MUST ensure that the templates are identical to the current upstream version on Github.

Step 2. Processed MiAIRR data submission to GenBank/TLS

Processed sequence data will be submitted to the "Targeted Locus Study" (TLS) section of GenBank. The details of this submission process are currently still being finalized. Basically the procedure is identical to a conventional GenBank submission with the exception of additional keywords marking it as TLS submission.

Non-productive records should be removed before the data submission or use an alternative annotation as described in the specification document.

- Generating MiAIRR compliant GenBank/TLS submissions: https://changeo.readthedocs.io/en/stable/examples/genbank.html

GenBank provides multiple tools (GUI and command-line) to submit data:

- BankIt, a web-based submission tool with wizards to guide the submission process
- Sequin, NCBI's stand-alone submission tool with wizards to guide the submission process is available by FTP for use on for Windows, macOS and Unix platforms.
- Tbl2asn is the recommended tool for the bulk data submission. It is a command-line program that automates the creation of sequence records files (.sqn) for submission to GenBank, driven by multiple tabular unput data files. Documentation and download options can be found under https://www.ncbi.nlm.nih.gov/genbank/tbl2asn2/.

MiAIRR-to-NCBI Specification

Outline of INSDC reporting procedure

TODO: Outline the reporting procedure for data sets 1-4

In terms of standard compliance it is currently REQUIRED[1] to deposit information for MiAIRR data sets 5 and 6 in general-purpose sequence repositories for which an AIRR-accepted specification on information mapping MUST exist. However, users should note that in the future additional AIRR-sanctioned mechanisms for data deposition will become available as specified by the AIRR Common Repository Working Group. The mapping of data items in MiAIRR data sets 5 and 6 differs substantially in size and structure and therefore requires distinct reporting procedures:

- Set 5: This is free text information describing the work flow, tools and parameters of the sequence read processing. It is REQUIRED that this information is deposited as a freely available document, permanently linked via a DOI. Note that is currently neither a specific format for this document nor a recommended service provider for obtaining the DOI.
- Set 6: This is specified to contain the consensus sequence and the following information obtained from the initial analysis: V, D and J segment, C region and IMGT-JUNCTION[2] [LIGMDB_V12]. These will be deposited in a general-purpose INSDC repository, using the record structure described below.

INSDC records were originally designed to hold individual Sanger sequences. Therefore each record will contain a header with information largely identical between all records in an AIRR sequencing study. Records can be concatenated for uploading.

The INSDC feature table (FT) [INSDC_FT] is a sequence annotation standard used within the INSDC records and assigns information to specified positions on the reported sequence string. In regard to the correct location of the provided annotation, it should especially be noted that some V(D)J inference tools will return coordinates referring to the reference instead of the query sequence. As the sequence submitted in a record MUST be identical to the query sequence, the positions provided by the V(D)J inference tool MUST, if necessary, be translated back onto the query sequence. In case the start and/or end of a feature cannot be reliably determined or is not present in the reported sequence[3], open intervals CAN be used for reporting. However, open intervals MUST NOT be used to deliberately obfuscate known positions.

In addition to the required information specified in *Table_1*, users CAN use all valid FT keys/qualifiers to provide further annotation for the reported sequences. However, a record MUST still be compliant with this specification, if such OPTIONAL information would be removed, meaning that it is FORBIDDEN to move REQUIRED information into OPTIONAL keys/qualifiers. In addition, users MUST NOT use keys/qualifiers that could create ambiguity with the keys/qualifiers specified here.

[1] See the "Glossary" section on how to interpret term written in all-caps.

[2] Note that according to IMGT definition this is a superset of the CDR3.

[3] This can occur e.g. in paired-end sequencing of head-to-head concatenated transcripts, where the 5' end of the V segment is present in the amplicon, but cannot be precisely determined.

element	FT key	FT qualifier	FT value	REQUIRED (if used by original study)
V segment	`V_segment`	`/gene`	see [Feature table]	yes
D segment	`D_segment`	`/gene`	see [Feature table]	yes; if *IGH*, *TRB* or *TRD* sequence
J segment	`J_segment`	`/gene`	see [Feature table]	yes
C region	`C_region`	`/gene`	see [Feature table]	yes
JUNCTION	`CDS`	`/function`	"JUNCTION"	yes

Table 1: Summary of the mapping of mandatory AIRR MiniStd data set 6 elements to the INSDC feature table (FT). Note that the overall record will contain additional information, such as cross-references linking the deposited sequence reads and metadata.

Element mapping

The broad strategy of element mapping to the various repositories is depicted in *Table_2*.

MiAIRR data set / subset	target repository
1 / study	BioProject
1 / subject	
1 / diagnosis & treatment	
2 / sample	BioSample
3 / processing (cells)	
3 / processing (nucleic acids)	SRA
4 / raw sequences	
5 / processing (data)	user-defined DOI
6 / Processed sequences & annotations	Genbank

Table 2: Summary of the mapping of MiAIRR data sets to the various repositories

Mapping of data sets 1-4 to BioProject/BioSample/SRA

TODO: Include item-by-item mapping [NCBI_NBK47528]

Mapping of data set 5 to a user-defined repository

While several mandatory item have been defined in this data set, there is currently no mapping as the reporting procedure is implemented as a free text document. AIRR RECOMMENDS to use Zenodo for deposition of these documents, as it is hosted by CERN and supports versioned DOIs (termed "concept" DOI). Users SHOULD use the existing `AIRR` tag when submitting documents to increase the visiblity of their study.

Mapping of data set 6 to INSDC

Users should note that while the FT is standardized, the overall sequence record structure diverges between the three INSDC repositories. The following section refers to items at or above the hierarchy level of the FT using the GenBank specification [GENBANK_FF], the corresponding designations of ENA [ENA_MANUAL] are provided in parenthesis[11].

[11] Note that there is currently no submission specification for ENA. This information is provided for reference only and will be moved to a separate document in the future.

Record header

The header MUST contain all of the following elements:

- REQUIRED: header structure as specified by the respective INSDC repository [ENA_MANUAL] [GENBANK_FF] [GENBANK_SR].
- FORBIDDEN: The `DEFINITION` entry will be autopopulated by information provided in the FT part (`misc_feature`, `/note`).
- REQUIRED: identifier of the associated SRA record (MiAIRR data set 4) as `DBLINK` (ENA: `DR` line). Note that it is **not** possible to refer to individual raw reads, only the full SRA collections can be linked.
- REQUIRED: in the `KEYWORDS` field (ENA: `KW` line):
  - the term "TLS"
  - the term "Targeted Locus Study"
  - the term "AIRR"
  - the term "MiAIRR:<x>.<y>" with <x> and <y> indicating the used version and subversion of the MiAIRR standard.
- REQUIRED: DOI of the associated free-text record containing the information on data processing (MiAIRR data set 5) as `REMARK` within a `REFERENCE`[4] (ENA: `RX` line).
- OPTIONAL: The use of structured comments is currently evalutated for use in future versions of the MiAIRR standard.

Feature table

The feature table, indicated by `FEATURES` (ENA: `RX` line), MUST or SHOULD contain the following keys/qualifiers:

General sequence information

- REQUIRED: key `source` containing the following qualifiers:
  - REQUIRED: qualifier `/organism` (required by [INSDC_FT]).
  - REQUIRED: qualifier `/mol_type` (required by [INSDC_FT]).
  - REQUIRED: qualifier `/citation` pointing to the reference in the header (`REFERENCE`, ENA: `RN` line) that links to the data set 5 document.
  - REQUIRED: qualifier `/rearranged`[5].
  - REQUIRED: qualifier `/note` containing the `AIRR_READ_COUNT` keyword to indicate the read number used for the consensus. The criteria for selecting these reads and the procedure used to build the consensus SHOULD be reported as part of data set 5.
  - OPTIONAL: qualifier `/note` containing the `AIRR_INDEX_CELL` keyword for single-cell experiments. The value of the keyword SHOULD only contain alpha-numeric characters and MUST be identical for sequences derived from the same cell of origin.

[4] The current GenBank record specification does not include a separate key for DOIs.

[5] Although FT does specify a */germline* qualifier for non-rearranged sequences it has not been included in this specification as there is no obvious use case for it. In addition, non-rearranged transcripts would lack a number of other features that are assumed to be present, first of all the JUNCTION.

  - RECOMMENDED: qualifiers `/assembly_gap` and `/linkage_evidence` to annotate non-overlapping paired-end sequences.
  - RECOMMENDED: qualifier `/strain`, if `/organism` is "Mus musculus".

Note that additional qualifiers might be REQUIRED by GenBank to harmonize the GenBank record with the BioSample referenced by it in the header. A list of known BioSample keyword and GenBank qualifiers that MUST contain the same information can be found below. Whether (and in which direction) the existence of a keyword/qualifiers triggers a requirement in the corresponding record is currently unknown. Please report any undocumented requirements surfacing during submission to the MiAIRR team.

BioSample keyword	GenBank FT qualifier
`cell type`	`/cell_type`
`isolate`	`/isolate`
`sex`	`/sex`
`tissue`	`/tissue_type`

Segment and region annotation

The following keys MUST be used for annotation according to their FT definition, if the respective item has been reported by the original study:

- REQUIRED: key `V_region`. Note that this key MUST NOT be used to annotate V segment leader sequence[6][7].
- REQUIRED: key `misc_feature` with coordinates identical to those given in `V_region`. This key MUST contain a `/note` qualifier that contains a string as value, which describes the general type of variable region described by the record. The string MUST match the regular expression

```
/^(immunoglobulin (heavy|light)|T cell receptor (alpha|beta|gamma|delta)) chain␣
↪variable region$/
```

  This string will be used as record heading upon import into Genbank. Note that while this behavior of Genbank is undocumented, the procedure has been approved by NCBI.

- REQUIRED: key `V_segment`, both coordinates MUST be within `V_region`. Note that this key MUST NOT be used to annotate V segment leader sequence[6][7].
- REQUIRED: key `D_segment`, both coordinates MUST be within `V_region`. This key is only REQUIRED for sequences of applicable loci (*IGH*, *TRB*, *TRD*[8]). In the rare case of rearrangements using two D segments, this key SHOULD occur twice, but the coordinates of both keys MUST NOT overlap.
- REQUIRED: key `J_segment`, both coordinates MUST be within `V_region`.
- REQUIRED: key `C_region`, both coordinates MUST NOT overlap with `V_region`. If the region can be unambiguously identified, the respective official gene symbol MUST be reported using the `/gene` qualifier. If only the isotype (e.g. IgG) but not the subclass (e.g. IgG1) can be identified, a truncated gene symbol (e.g. IGHG instead of IGHG1) SHOULD be reported instead[9].

Each `[VDJ]_segment` key MUST or SHOULD contain the following qualifiers:

[6] The FT explicitly states that *V_segment* does **not** cover the leader sequence. The definition of *V_region* is slightly more ambiguous, however in combination with the *V_segment* definition, it becomes clear that the leader is also not considered to be a part of *V_region*. Therefore the leader sequence should be implicitly annotated as the region between the start of *CDS* and the start of *V_region*.

[7] Previously the leader was implicitly annotated as the region between *CDS* start and *V_region* start. As it was decided to drop the "global" CDS to make it easier to accommodate for INDELs, this is currently not an option anymore.

[8] For simplicity, this document only uses human gene symbols. For non-human species the specification pertains to the respective orthologs.

[9] This approach has been approved by NCBI.

- REQUIRED: qualifier /gene, containing the designation of the inferred segment, according to the database in the first /db_xref entry. This qualifier MUST NOT contain any allele information.
- RECOMMENDED: qualifier /allele, containing the designation of the inferred allele, according to the database in the first /db_xref entry. Note that while INSDC does not specify any format for this qualifier, AIRR compliance REQUIRES that this field only contains the allele string, i.e. without the gene name or separator characters.
- REQUIRED: qualifier /db_xref, linking to the reference record of the inferred segment in a germline database [INSDC_XREF]. This qualifier can be present multiple times, however only the first entry is mandatory and MUST link to the database used for the segment designation given with /gene and (if present) /allele.

  Note on referencing IMGT databases: There are two IMGT database available in the controlled vocabulary [INSDC_XREF]:

  - IMGT/GENE-DB: This is the genome database, which requires that a reference sequence has been mapped to genomic DNA. When using this database as reference, note that you can only refer to the gene symbol **not** the allele. In the case of ambiguous allele calls (see below) this means that you MUST NOT annotate any /allele at all. Nevertheless, this SHOULD be the default database for applications using IMGT as reference, as the sequence for each gene/allele is unique.
  - IMGT/LIGM: This database collects sequences described in INSDC databases (GenBank/ENA/DDBJ). As it might contain multiple entries representing a given gene/allele, it is NOT RECOMMENDED to use it unless that inference gene/allele is only present in IMGT/LIGM and not in IMGT/GENE-DB.
- RECOMMENDED: /inference to indicate the tool used for segment inference. The description string SHOULD use COORDINATES as category and aligment as type [INSDC_FT].

Annotation of sequences producing multiple hits with identical scores is problematic and is ultimately at the discretion of the depositing researcher. However, the algorithms used for tie-breaking SHOULD be documented in data set 5. In addition, the following procedures MUST be followed:

- Certain gene, ambiguous allele: If multiple alleles of the same gene match to the sequence, the /allele qualifier MUST NOT be used. As the REQUIRED /db_xref qualifier will ofter refer to a specific allele, all equal hits SHOULD be annoted via this qualifier (which can be use multiple times). Also see the note on the limitations of the IMGT/GENE-DB reference database above.
- Ambiguous gene: Pick one, annotate using the qualifiers as noted for ambiguous allele.

JUNCTION annotation

INSDC does currently not define a key to annotate JUNCTION[10]. Therefore the following procedure MUST be used:

- REQUIRED: key CDS, indicating the positions of

  1. the first bp of the first AA of JUNCTION
  2. the last bp of the last AA of JUNCTION as determined by the utilized V(D)J inference tool.

  Open coordinates MUST be used for both coordinates to allow for automated creation of the /translated qualifier providing the peptide sequence. Further note that a non-productive JUNCTION can have a length not divisible by three. This key contains the following qualifiers:

  - REQUIRED: qualifier /codon_start with the assigned value “1”.
  - REQUIRED: qualifier /function with the assigned value “JUNCTION”.
  - REQUIRED: qualifier /product with an assigned value matching the regular expression

[10] NCBI confirmed that once there would be enough datasets using the *JUNCTION* tag as specified here, a motion for an INSDC-sanctioned key could be initiated.

```
/^(immunoglobulin (heavy|light)|T cell receptor (alpha|beta|gamma|delta))␣
↪chain junction region$/
```

The variable region referred to in the string MUST be the same as the one given in the `misc_feature` key.

- RECOMMENDED: qualifier `/inference`, indicating the tool used for positional inference. The description string SHOULD use `COORDINATES` as category and `protein motif` as type [INSDC_FT].
- FORBIDDEN: qualifier `/translated`, which will be automatically added by Genbank.

Note that the complete `CDS` key will be removed by Genbank if the translation contains stop codons or to many "N" (exact number unknown). As such a record will lack a central piece of REQUIRED information it is RECOMMENDED that submitters either

- remove the complete record or
- replace the `CDS` with a `misc_feature` key while at the same time removing the `/codon_start` and `/product` qualifiers

upfront, as described in the submission manual. If the submitter chooses the replacement option, it has to be ensured that the annotated coordinates are actually valid and not affect by the frame- shift.

Record body

The record body starts after `ORIGIN` (ENA: `SQ` line) and MUST contain:

- the consensus sequence

References

Footnotes

Appendix

Example record (GenBank format)

```
LOCUS       AB123456                 420 bp    mRNA    linear   EST 01-JAN-2015
DEFINITION  TLS: Mus musculus immunoglobulin heavy chain variable region,
            sequence.
ACCESSION   AB123456
VERSION     AB123456.7
KEYWORDS    TLS; Targeted Locus Study; AIRR; MiAIRR:1.0.
SOURCE      Mus musculus
  ORGANISM  Mus musculus
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;
            Euteleostomi; Mammalia; Eutheria; Euarchontoglires; Glires;
            Rodentia; Sciurognathi; Muroidea; Muridae; Murinae; Mus.
REFERENCE   1  (bases 1 to 420)
  AUTHORS   Stibbons,P.
  TITLE     Section 5 information for experiment FOO1
  JOURNAL   published (01-JAN-2000) on Zenodo
  REMARK    DOI:10.1000/0000-12345678
REFERENCE   2  (bases 1 to 420)
```

(continues on next page)

(continued from previous page)

```
  AUTHORS   Stibbons,P.
  TITLE     Direct Submission
  JOURNAL   Submitted (01-JAN-2000) Center for Transcendental Immunology,
            Unseen University, Ankh-Morpork, 12345, DISCWORLD
DBLINK      BioProject: PRJNA000001
            BioSample: SAMN000001
            Sequence Read Archive: SRR0000001
FEATURES             Location/Qualifiers
     source          1..420
                     /organism="Mus musculus"
                     /mol_type="mRNA"
                     /strain="C57BL/6J"
                     /citation=[1]
                     /rearranged
                     /note="AIRR_READ_COUNT:123”
     V_region        1..324
     misc_feature    1..324
                     /note="immunoglobulin heavy chain variable region"
     V_segment       1..257
                     /gene="IGHV1-34"
                     /allele="01"
                     /db_xref="IMGT/LIGM:AC073565"
                     /inference="COORDINATES:alignment:IgBLAST:1.6"
     D_segment       266..272
                     /gene="IGHD2-2"
                     /allele="01"
                     /db_xref="IMGT/LIGM:AJ851868"
                     /inference="COORDINATES:alignment:IgBLAST:1.6"
     J_segment       291..324
                     /gene="IGHJ4"
                     /allele="01"
                     /db_xref="IMGT/LIGM:V00770"
                     /inference="COORDINATES:alignment:IgBLAST:1.6"
     CDS             <258..>290
                     /codon_start=1
                     /function="JUNCTION"
                     /product="immunoglobulin heavy chain junction region"
                     /inference="COORDINATES:protein motif:IgBLAST:1.6"
                     /translated="CARAGVYDGYTMDYW"
     C_region        325..420
                     /gene="Ighg2c"
ORIGIN
        1 agcctggggc ttcagtgaag atgtcctgca aggcttctgg ctacacattc actgactata
       61 acatacactg ggtgaagcag agccatggaa agagccttga gtggattgca tatattaatc
      121 ctaacaatgg tggttatggc tataacgaca agttcaggga caaggccaca ttgactgtcg
      181 acaggtcatc caacacagcc tacatggggc tccgcagcct gacctctgag gactctgcag
      241 tctattactg tgcaagagcg ggagtttacg acggatatac tatggactac tggggtcaag
      301 gaacctcagt caccgtctcc tcagccaaaa caacagcccc atcggtctat ccactggccc
      361 ctgtgtgtgg aggtacaact ggctcctcgg tgactctagg atgcctggtc aagggcaact
//
```

Glossary

- MUST / REQUIRED: Indicates that an element or action is necessary to conform to the standard.
- SHOULD / RECOMMENDED: Indicates that an element or action is considered to be best practice by AIRR,

but not necessary to conform to the standard.

- CAN / OPTIONAL: Indicates that it is at the discretion of the user to use an element or perform an action.
- MUST NOT / FORBIDDEN: Indicates that an element or action will be in conflict with the standard.

Abbreviations

- AA: amino acid
- bp: base pair
- DOI: digital object identifier
- FT: INSDC Feature Table
- INSDC: International Nucleotide Sequence Database Collaboration
- SRA: sequence read archive

Introduction

The MiAIRR standard

The MiAIRR standard (minimal information about adaptive immune receptor repertoires) is a minimal reporting standard for experiments using sequencing-based technologies to study adaptive immune receptors (e.g. T cell receptors or immunoglobulins). It is developed and maintained by the Minimal Standards Working Group of the Adaptive Immune Receptors Repertoire (AIRR) Community [Breden_2017]. The current version (1.0) of the standard has been recently published [Rubelt_2017] and was passed by the general assembly at the annual AIRR Community meeting in December 2017. MiAIRR requires researchers to report six sets of information:

1. study, subject, diagnosis & intervention
2. sample collection
3. sample processing and sequencing
4. raw sequencing data
5. data processing
6. processed sequences with a basic analysis results

However, MiAIRR only describes the mandatory data items that have to be reported, but neither provides details how and where to deposit data nor specifies data types and formats. Therefore this document aims to provide both a submission manual for users as well as a detailed data specification for developers.

Requirement Levels of AIRR Schema Fields

Clarification of Terms

- The terms “MUST”, “MUST NOT”, “REQUIRED”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY” and “OPTIONAL” are to be interpreted as described in [RFC2119].
- The terms “IF” and “ONLY IF” are are to be interpreted as sufficent and necessary requirement, respectively.

- The term "NULL-LIKE" is an extension of the `NULL` term in SQL and its equivalents in other programming languages, referring to the absence of data in a field (i.e., the field is empty). NULL-LIKE **additionally** includes the following terms, which also define the reason why the information is missing. As these terms are expected to be provided as text, the field would not be `NULL` but nevertheless NULL-LIKE (i.e., it lacks biologically interpretable information).
  - `not_applicable`: There is no meaningful value for this field due to study design (e.g., `sex` for a phage library).
  - `not_collected`: Data for this field was not collected during the study.
  - `missing`: Data for field was collected, but is not available now.

Categories of AIRR Schema Fields

- Fields MUST be indicated by the `x-airr:miairr` property IF and ONLY IF the field or its content is governed by the MiAIRR data standard [Rubelt_2017].
- The `x-airr:miairr` property MUST be assigned to one of the following three requirement levels:
  - `essential`: Information on this field MUST be provided and is considered critical for the meaningful interpretation of the data. Therefore the value of such a field MUST NOT be NULL-LIKE. Due to this strict requirement, this level is only assigned to a small set of fields. Importantly, fields are **not** elevated to this level based on the fact that the respective information should typically be available to the data generator. This was decided to simplify MiAIRR-compliant data annotation by third parties, who might perform this task based on publicly available information only.
  - `important`: Information for this field MUST be provided. However, the field MAY be assigned a NULL-LIKE value if the respective information is not available. The majority of fields governed by the MiAIRR data standard are assigned to this level.
  - `defined`: Information for this field MAY be provided. However, IF information matching the semantic definition of the field is provided, this field MUST be used for reporting.

Compliance with the MiAIRR Data Standard

- Compliance to the MiAIRR Data Standard is currently a binary state, i.e., a data either is or is not compliant, there are not "grades" of compliance. However, additional requirements for specific use cases might be defined in the future.
- Data sets are considered MiAIRR-compliant ONLY IF all `essential` and `important` fields are reported.
- Note that `important` fields with NULL-LIKE values MUST NOT be dropped from a data set.
- Implementors of data entry interfaces SHOULD NOT set the default value of `important` fields to NULL-LIKE values, i.e., users should be required to actively select the values.

Metadata Annotation Guidelines

Purpose of this Document

This document describes the RECOMMENDED ways to provide metadata annotation for various experimental setups.

Clarification of Terms

- The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Individual fields

library_generation_method

The library_generation_method describes how the nucleic acid annotated in template_class that encodes the V(D)J-rearrangement it reverse-transcribed, amplified and/or otherwise prepared for further processing. Typically this procedure will precede further NGS platform- specific steps, however these procedures MAY be combined. The field uses a controlled vocabulary, the individual values are described below:

template_class	library_generation_method	Methodology
DNA	PCR	Conventional PCR on genomic DNA of a vertebrate host (requires: synthetic == false)
		Conventional PCR on DNA of a synthetic library (requires: synthetic == true)
RNA	RT(RHP)+PCR	RT-PCR using random hexamer primers
	RT(oligo-dT)+PCR	RT-PCR using oligo-dT primers
	RT(oligo-dT)+TS+PCR	5'-RACE PCR (i.e. RT is followed by a template switch (TS) step) using oligo-dT primers
	RT(oligo-dT)+TS(UMI)+PCR	5'-RACE PCR using oligo-dT primers and template switch primers containing unique molecular identifiers (UMI), i.e., the 5' end is UMI-coded
	RT(specific)+PCR	RT-PCR using transcript-specific primers
	RT(specific)+TS+PCR	5'-RACE PCR using transcript- specific primers
	RT(specific)+TS(UMI)+PCR	5'-RACE PCR using transcript- specific primers and template switch primers containing UMIs
	RT(specific+UMI)+PCR	RT-PCR using transcript-specific primers containing UMIs (i.e., the 3' end is UMI-coded)
	RT(specific+UMI)+TS+PCR	5'-RACE PCR using transcript- specific primers containing UMIs (i.e., the 3' end is UMI-coded)
	RT(specific)+TS	RT-based generation of dsDNA **without** subsequent PCR. This is used by RNA-seq kits.
any	other	Any methodology not covered above

Specific Use Cases and Experimental Setups

Synthetic libraries

In synthetic libraries (e.g. phage or yeast display), particles present genetically engineered constructs (e.g. scFv fusion receptors) on their surface. As this deviates substantially from other workflows, the following annotation SHOULD/MUST be used:

- In general, Subject should be interpreted as the initial library that undergoes a mutation/selection procedure.
- synthetic: MUST be set to true
- species: It is assumed that every synthetic library is derived from V and J genes that exist in some vertebrate species. This field SHOULD encode this species. Importantly, it MUST NOT encode the phage vector, the

bacterial host or the comparable biological component of the library system that constitutes the presenting particle.

- `sample_type`: SHOULD be `NULL`.
- `single_cell`: Only `true` if individual particles are isolated and sequenced. Note that colonies or plaques, even if containing genetically identical particles, *per se* do not match this definition and therefore MUST be annotated as `false`.
- `cell_storage`: SHOULD be used for non-cellular particles analogously.
- `physical_linkage`: For scFv constructs the `hetero_prelinkeded` term MUST be used. VHH (i.e. camelid) libraries SHOULD annotate `none` as there is only a single rearrangement envolved.

2.3.2 AIRR Data Representations

AIRR Data Representations are versioned specifications that consist of a file format and a well-defined schema. The schema is provided in a machine-readable YAML document that follows the OpenAPI v2.0 specification. The schema defines the data model, field names, data types, and encodings for AIRR standard objects. Strict typing enables interoperability and data sharing between different AIRR-seq analysis tools and repositories, and some fields use a controlled vocabulary or an ontology for value restriction. Specification extensions are utilized to define AIRR-specific attributes.

FAIR Principles

We desire AIRR standard objects to be FAIR (findable, accessible, interoperable and reusable) [Wilkinson_2016]:

- findable: by giving AIRR standard objects a globally unique identifier
- accessible: by providing an API where AIRR standard objects can be queried and downloaded
- interoperable: by defining a OpenAPI schema for the AIRR standard objects
- reusable: by linking the AIRR standard objects together into a standard formats

AIRR Data Model

The MiAIRR standard defines the minimal information for submission and publication of AIRR-seq datasets. The standard defines a set of data elements for this information and organizes them into six high-level sets.

- Study, Subject and Diagnosis
- Sample Collection
- Sample Processing and Sequencing
- Raw Sequences
- Data Processing
- Processed Sequences with Annotations

However beyond these sets, MiAIRR does not define any structure, data model or relationship between the data elements. This provides flexibility for the information to be stored in various database repositories but is problematic for interoperability and reusability of that information by computer programs. The AIRR Data Model overcomes these issues by defining a schema for the MiAIRR data elements, structuring them within schema objects, defining the relationship between those objects, and defining a file format.

Here are the primary schema objects of the AIRR Data Model:

Schema Object	Description
`Study`	Information about the experimental study design, including the title of the study, laboratory contact information, funding, and linked publications.
`Subject`	Information about the study cohorts and individual subjects, including species, sex, age, and ancestry.
`Diagnosis`	Information about disease state(s), therapies, and study group membership (e.g., control versus disease).
`Sample`	Information about the origin and expected composition of the biological sample(s). This set aims to capture essential information about the collection of a sample, including its source (e.g., anatomical site), its provenance (provider), and the experimental condition (e.g., the time point during the course of a disease or treatment).
`CellProcessing`	Information about the cell subset being profiled, as defined by the investigator, and the flow cytometry or other markers used to select the subset. Additional information includes the number of cells per sample and whether cells were prepared in bulk or captured as single cells.
`NucleicAcidProcessing`	Information about nucleic acid sample type (e.g., RNA versus DNA) and how immune-receptor gene rearrangements were amplified and sequenced (for example, RACE-PCR versus multiplex PCR, paired PCR, and/or varying read length and sequencing chemistries).
`SequencingRun`	Information about the sequencing run, such as the number of reads, read lengths, quality control parameters, the sequencing kit and instrument(s) used, and run batch number. Also includes information about the raw data for the sequencing run (e.g., FASTQ files).
`DataProcessing`	Information about the data processing to transform the raw sequencing data into `Rearrangements`.
`Repertoire`	Composite object that combines the schema objects `Study`, `Subject`, `Diagnosis`, `Sample`, `CellProcessing`, `NucleicAcidProcessing`, `SequencingRun`, and `DataProcessing`. Each `Repertoire` has a unique identifier `repertoire_id` for linking with other data files, e.g. `Rearrangements`. `Repertoires` have their own schema and file format described *here*.
`Rearrangement`	Annotated sequences describing adaptive immune receptor chains. `Rearrangements` have their own schema and file format described *here*.

Relationship between Schema Objects

The MiAIRR categories are hierarchical, and includes information about the study, the subjects, the collected samples and how they are processed, details of the sequencing protocol, and information about the data analysis. The top-down relationships are either 1-to-n indicating the top level object can be related to any number of sub-level objects, or n-to-n indicating any number of top level object can be related to any number of sub-level objects. Lastly, 1-to-1 indicates the top level object is related to a single sub-level object.

- `Study` 1-to-n with `Subject`. A study may contain any number of subjects.
- `Subject` 1-to-n with `Diagnosis`. Each subject may contain any number of diagnoses.
- `Subject` 1-to-n with `Sample`. Each subject may contain any number of samples.
- `Sample` 1-to-n with `CellProcessing`. A sample may have any number of cell processing records.
- `CellProcessing` 1-to-n with `NucleicAcidProcessing`. A cell processing record may have any number of nucleic acid processing records.
- `NucleicAcidProcessing` 1-to-n with `SequencingRun`. A nucleic acid processing records may have any number of sequencing runs.
- `SequencingRun` n-to-n with `DataProcessing`. Multiple sequencing runs can be combined in a data processing, and multiple data processing can be done on a sequencing run.

However, this hierarchy is deep and complicated. Therefore to simplify the processing of this information, we denormalized the hierarchy around the conceptual `Repertoire` object. This denormalization represents many relationships as 1-to-1 which simplifies the structure. A single `Repertoire` has these relationships with the primary schema objects.

- `Repertoire` 1-to-1 with `Study`. A repertoire is for a single study, though a study may have multiple repertoires.
- `Repertoire` 1-to-1 with `Subject`. A repertoire is for a single subject, though a subject may have other repertoires defined.
- `Sample` 1-to-1 with `CellProcessing`, `NucleicAcidProcessing`, and `SequencingRun`. A sample is associated with a single chain of sample processing from initial collection, through cell and nucleic acid processing, to sequencing.
- `Repertoire` 1-to-n with `Sample`. Generally a repertoire has a single sample, but sometimes studies perform technical replicates or re-sequencing to generate additional data, and these studies will have multiple samples, which are to be combined and analyzed together as part of the same repertoire.
- `Repertoire` 1-to-n with `DataProcessing`. A repertoire can be analyzed multiple times. More details about multiple data processing is provided below.

The trade-off with denormalization of the hierarchy is that it causes duplication of data. For example, two repertoires for the same study will have the `Study` information duplicated within each of the two repertoire records; likewise multiple repertoires for the same subject will have the `Subject` information duplicated.

While the denormalized `Repertoire` simplifies read-only access to the MiAIRR information, it complicates data entry and write access to the information because updates need to be propagated to all of the duplicate records. Therefore, `Repertoire` was designed to be easily transformed into a normalized form, representing the full hierarchy of the objects, by utilizing the *study_id*, *subject_id*, and *sample_id* fields to uniquely identify the `Study`, `Subject` and `Sample` objects across multiple repertoires. The exception is that `CellProcessing` and `NucleicAcidProcessing` do not have their own unique identifiers, so they are included within `Sample`.

AIRR extension properties

The OpenAPI V2 specification provides the ability to define extension properties on schema objects. These are additional properties on the schema definition directly, not to be confused with additional properties on the data. These extension properties allow those schema definitions to be annotated with MiAIRR and AIRR specific information. Instead of creating separate extensions for each property, a single extension `x-airr` property is defined, which is an object that contains any number of properties. Within the AIRR schema, `AIRR_Extension` defines the schema for the `x-airr` object and the properties within it. Here is a list of the currently supported AIRR extension properties:

Extension	Description
`miairr`	Present if the annotated property is a MiAIRR data standard element. Always has a *requirement level* assigned to it.
`nullable`	Assumes `miairr`. False if the annotated property must not be `NULL` by the MiAIRR standard, otherwise True or null.
`set`	Assumes `miairr`. The MiAIRR set for the annotated property.
`subset`	Assumes `miairr`. The MiAIRR subset for the annotated property.
`name`	Assumes `miairr`. The MiAIRR field name.
`format`	Describes the format for the annotated property. Value is either `free text`, `controlled vocabulary` or `ontology`.
`ontology`	If `format=ontology` then this provides additional information about the ontology including draft status, name, URL and top node term.

Schema Definitions

Repertoire Schema

A Repertoire is an abstract organizational unit of analysis that is defined by the researcher and consists of study metadata, subject metadata, sample metadata, cell processing metadata, nucleic acid processing metadata, sequencing run metadata, a set of raw sequence files, data processing metadata, and a set of Rearrangements. A Repertoire gathers all of this information together into a composite object, which can be easily accessed by computer programs for data entry, analysis and visualization.

A Repertoire is specific to a single subject otherwise it can consist of any number of samples (which can be processed in different ways), any number of raw sequence files, and any number of rearrangements. It can also consist of any number of data processing metadata objects that describe the processing of raw sequence files into Rearrangements.

Typically, a Repertoire corresponds to the biological concept of the immune repertoire for that single subject which the researcher experimentally measures and computationally analyzes. However, researchers can have different interpretations about what constitutes the biological immune repertoire; therefore, the Repertoire schema attempts to be flexible and broadly useful for all AIRR-seq studies.

Another researcher can take the same raw sequencing data and associated metadata and create their own Repertoire that is different from the original researcher's. A common example is to define a repertoire that is a subset such as "productive rearrangements for IGHV4" whereas the original researcher defined a more generic "B cell repertoire". This new Repertoire would have much of the same metadata as the original Repertoire, except associated with a different study, and with additional information in the data processing metadata that describes how the rearrangements were filtered down to just the "productive rearrangements for IGHV4". Likewise, another researcher may get access to the original biosample material and perform their own sample processing and sequencing, which also would be a new Repertoire. That new Repertoire could combine samples from the original researcher's Repertoire with the new sample data as a large dataset for the subject.

Multiple Data Processing on a Repertoire

Data processing can be a complicated multi-stage process. Documenting the process in a formal way is challenging because of the diversity of actions that may be performed. The MiAIRR standard requires documentation of the process but in an informal way with free text descriptions. A Repertoire might undergo multiple different data processing for any number of reasons, e.g. to compare the results from different toolchains, or to compare different settings for the same toolchain.

It is expected that all of the Samples of a Repertoire will be processed together within a DataProcessing. That is, a DataProcessing that only uses some but not all samples in a Repertoire could be confusing to users and appear as though data is missing. Likewise, processing some samples within a Repertoire with one DataProcessing and the remaining samples with a different DataProcessing could also confuse users. Because DataProcessing is unstructured information, it is not possible to validate that all Samples in a Repertoire are being processed together, so this expectation cannot be strictly enforced.

Having multiple DataProcessing for a Repertoire will create multiple sets of Rearrangements that are distinct and separate from each other. Analysis tools need to be careful not to mix these sets of Rearrangements from different DataProcessing because it can generate incorrect results. The identifier data_processing_id was added so Rearrangements can identify their specific DataProcessing.

Linking Data

Each Repertoire has a unique repertoire_id identifier. This identifier should be globally unique so that repertoires from multiple studies can be combined together without conflict. The repertoire_id is used to link other AIRR data to a Repertoire. Specifically, the *Rearrangements Schema* includes repertoire_id for referencing the specific Repertoire for that Rearrangement.

If a Repertoire has multiple DataProcessing then data_processing_id should be used to distinguish the appropriate DataProcessing within the Repertoire. The Rearrangements contains data_processing_id for this purpose. The data_processing_id is only unique within a Repertoire so repertoire_id should first be used to get the appropriate Repertoire object and then data_processing_id used to acquire the appropriate DataProcessing.

It is expected that typical Repertoires might only have a single DataProcessing, in which case repertoire_id and data_processing_id will be semantically equivalent and only the former should be used.

If a Repertoire has multiple sample processing objects in the sample array then sample_processing_id should be used to distinguish the the approrpiate sample processing object within the Repertoire. The Rearrangement object can contain a sample_processing_id to uniquely identify a sample processing object within a Repertoire. Like data_processing_id, the sample_processing_id is only unique within the Repertoire so repertoire_id should first be used to get the appropiate Repertoire object and then sample_processing_id should be used to determine the appropiate sample processing object that is associated with the Rearrangement. If the Rearrangement object does not have a sample_processing_id then it can be assumed that the rearrangement is associated with all of the samples in the Repertoire (e.g. the rearrangement is a collapsed rearrangement across multiple samples).

It is expected that Repertoires might often have a single sample processing object, in which case repertoire_id and sample_processing_id will be semantically equivalent and only the former should be used.

Finally, if it is necessary to link a Rearrangement object with a unique pairing of sample processing and DataProcessing, the repertoire_id of the Rearrangement object should be used to identify the correct Repertoire object and then the data_processing_id should be used to identify the correct DataProcessing metadata and the sample_processing_id should be used to identify the correct sample processing metadata within that Repertoire.

Duality between Repertoires and Rearrangements

There is an important duality relationship between Repertoires and Rearrangements, specifically with the experimental protocols described in the Repertoire versus the annotations on Rearrangements. A Repertoire defines an experimental design for what a researcher intends to measure or observe, while the Rearrangements are what was actually measured and observed. Technically, the border between the two occurs at sequencing, that is when the biological physical entity (prepared DNA) is measured and recorded as information (nucleotide sequence).

This duality is important when considering how to answer certain questions. For example, locus for Rearrangements may have the value "IGH" which indicates that B cell heavy chain receptors were measured, yet the Repertoire might have "T cell" in cell_subset which indicates the researcher intended to measure T cells. This conflict between the two indicates something is wrong. Differences can occur in many ways, as with errors in the experimental protocol, or data processing might have incorrectly processed the raw sequencing data leading to invalid annotations.

File Format Specification

Files are YAML/JSON with a structure defined below. Files should be encoded as UTF-8. Identifiers are case-sensitive. Files should have the extension .yaml, .yml, or .json.

File Structure

- The file as a whole is considered a dictionary (key/value pair) structure with the keys Info and Repertoire.

- The file can (optionally) contain an Info object, at the beginning of the file, based upon the Info schema in the OpenAPI V2 specification. If provided, version in Info should reference the version of the AIRR schema for the file.
- The file should correspond to a list of Repertoire objects, using Repertoire as the key to the list.
- Each Repertoire object should contain a top-level key/value pair for repertoire_id that uniquely identifies the repertoire.
- Some fields require the use of a particular ontology or controlled vocabulary.
- The structure is the same regardless of whether the data is stored in a file or a data repository. For example, The *ADC API* will return a properly structured JSON object that can be saved to a file and used directly without modification.

Repertoire Fields

Download as TSV

Name	Type	Attributes	Definition
repertoire_id	string	optional, identifier, nullable	Identifier for the repertoire object. This identifier should be globally unique so that repertoires from multiple studies can be combined together without conflict. The repertoire_id is used to link other AIRR data to a Repertoire. Specifically, the Rearrangements Schema includes repertoire_id for referencing the specific Repertoire for that Rearrangement.
repertoire_name	string	optional, nullable	Short generic display name for the repertoire
repertoire_description	string	optional, nullable	Generic repertoire description
study	*Study*	required	Study object
subject	*Subject*	required	Subject object
sample	array	required	List of Sample objects
data_processing	array of *DataProcessing*	required	List of Data Processing objects

Study Fields

Download as TSV

Name	Type	Attributes	Definition
`study_id`	string	required, nullable	Unique ID assigned by study registry
`study_title`	string	required, nullable	Descriptive study title
`study_type`	*Ontology*	required, nullable	Type of study design
`study_description`	string	optional, nullable	Generic study description
`inclusion_exclusion_criteria`	string	required, nullable	List of criteria for inclusion/exclusion for the study
`grants`	string	required, nullable	Funding agencies and grant numbers
`collected_by`	string	required, nullable	Full contact information of the data collector, i.e. the person who is legally responsible for data collection and release. This should include an e-mail address.
`lab_name`	string	required, nullable	Department of data collector
`lab_address`	string	required, nullable	Institution and institutional address of data collector
`submitted_by`	string	required, nullable	Full contact information of the data depositor, i.e. the person submitting the data to a repository. This is supposed to be a short-lived and technical role until the submission is relased.
`pub_ids`	string	required, nullable	Publications describing the rationale and/or outcome of the study
`keywords_study`	array of string	required, nullable	Keywords describing properties of one or more data sets in a study

Subject Fields

`Download as TSV`

Name	Type	Attributes	Definition
subject_id	string	required, nullable	Subject ID assigned by submitter, unique within study
synthetic	boolean	required	TRUE for libraries in which the diversity has been synthetically generated (e.g. phage display)
species	*Ontology*	required	Binomial designation of subject's species
organism	*Ontology*	DEPRECATED	Binomial designation of subject's species
sex	string	required, nullable	Biological sex of subject
age_min	number	required, nullable	Specific age or lower boundary of age range.
age_max	number	required, nullable	Upper boundary of age range or equal to age_min for specific age. This field should only be null if age_min is null.
age_unit	*Ontology*	required, nullable	Unit of age range
age_event	string	required, nullable	Event in the study schedule to which *Age* refers. For NCBI BioSample this MUST be *sampling*. For other implementations submitters need to be aware that there is currently no mechanism to encode to potential delta between *Age event* and *Sample collection time*, hence the chosen events should be in temporal proximity.
age	string	DEPRECATED	
ancestry_population	string	required, nullable	Broad geographic origin of ancestry (continent)
ethnicity	string	required, nullable	Ethnic group of subject (defined as cultural/language-based membership)
race	string	required, nullable	Racial group of subject (as defined by NIH)
strain_name	string	required, nullable	Non-human designation of the strain or breed of animal used
linked_subjects	string	required, nullable	Subject ID to which *Relation type* refers
link_type	string	required, nullable	Relation between subject and *linked_subjects*, can be genetic or environmental (e.g.exposure)
diagnosis	array of *Diagnosis*	optional	Diagnosis information for subject

Diagnosis Fields

Download as TSV

Name	Type	Attributes	Definition
`study_group_description`	string	required, nullable	Designation of study arm to which the subject is assigned to
`disease_diagnosis`	*Ontology*	required, nullable	Diagnosis of subject
`disease_length`	string	required, nullable	Time duration between initial diagnosis and current intervention
`disease_stage`	string	required, nullable	Stage of disease at current intervention
`prior_therapies`	string	required, nullable	List of all relevant previous therapies applied to subject for treatment of *Diagnosis*
`immunogen`	string	required, nullable	Antigen, vaccine or drug applied to subject at this intervention
`intervention`	string	required, nullable	Description of intervention
`medical_history`	string	required, nullable	Medical history of subject that is relevant to assess the course of disease and/or treatment

Sample Fields

`Download as TSV`

Name	Type	Attributes	Definition
`sample_id`	string	required, nullable	Sample ID assigned by submitter, unique within study
`sample_type`	string	required, nullable	The way the sample was obtained, e.g. fine-needle aspirate, organ harvest, peripheral venous puncture
`tissue`	*Ontology*	required, nullable	The actual tissue sampled, e.g. lymph node, liver, peripheral blood
`anatomic_site`	string	required, nullable	The anatomic location of the tissue, e.g. Inguinal, femur
`disease_state_sample`	string	required, nullable	Histopathologic evaluation of the sample
`collection_time_point_relative`	string	required, nullable	Time point at which sample was taken, relative to *Collection time event*
`collection_time_point_reference`	string	required, nullable	Event in the study schedule to which *Sample collection time* relates to
`biomaterial_provider`	string	required, nullable	Name and address of the entity providing the sample

Tissue and Cell Processing Fields

`Download as TSV`

Name	Type	Attributes	Definition
`tissue_processing`	string	required, nullable	Enzymatic digestion and/or physical methods used to isolate cells from sample
`cell_subset`	*Ontology*	required, nullable	Commonly-used designation of isolated cell population
`cell_phenotype`	string	required, nullable	List of cellular markers and their expression levels used to isolate the cell population
`cell_species`	*Ontology*	optional, nullable	Binomial designation of the species from which the analyzed cells originate. Typically, this value should be identical to *species*, if which case it SHOULD NOT be set explicitly. Howver, there are valid experimental setups in which the two might differ, e.g. chimeric animal models. If set, this key will overwrite the *species* information for all lower layers of the schema.
`single_cell`	boolean	required, nullable	TRUE if single cells were isolated into separate compartments
`cell_number`	integer	required, nullable	Total number of cells that went into the experiment
`cells_per_reaction`	integer	required, nullable	Number of cells for each biological replicate
`cell_storage`	boolean	required, nullable	TRUE if cells were cryo-preserved between isolation and further processing
`cell_quality`	string	required, nullable	Relative amount of viable cells after preparation and (if applicable) thawing
`cell_isolation`	string	required, nullable	Description of the procedure used for marker-based isolation or enrich cells
`cell_processing_protocol`	string	required, nullable	Description of the methods applied to the sample including cell preparation/ isolation/enrichment and nucleic acid extraction. This should closely mirror the Materials and methods section in the manuscript.

Nucleic Acid Processing Fields

`Download as TSV`

Name	Type	Attributes	Definition
`template_class`	string	required	The class of nucleic acid that was used as primary starting material for the following procedures
`template_quality`	string	required, nullable	Description and results of the quality control performed on the template material
`template_amount`	string	required, nullable	Amount of template that went into the process
`library_generation_method`	string	required	Generic type of library generation
`library_generation_protocol`	string	required, nullable	Description of processes applied to substrate to obtain a library that is ready for sequencing
`library_generation_kit_version`	string	required, nullable	When using a library generation protocol from a commercial provider, provide the protocol version number
`pcr_target`	array of *PCRTarget*	optional	If a PCR step was performed that specifically targets the IG/TR loci, the target and primer locations need to be provided here. This field holds an array of PCRTarget objects, so that multiplex PCR setups amplifying multiple loci at the same time can be annotated using one record per locus. PCR setups not targeting any specific locus must not annotate this field but select the appropriate library_generation_method instead.
`complete_sequences`	string	required	To be considered *complete*, the procedure used for library construction MUST generate sequences that 1) include the first V gene codon that encodes the mature polypeptide chain (i.e. after the leader sequence) and 2) include the last complete codon of the J gene (i.e. 1 bp 5' of the J->C splice site) and 3) provide sequence information for all positions between 1) and 2). To be considered *complete & untemplated*, the sections of the sequences defined in points 1) to 3) of the previous sentence MUST be untemplated, i.e. MUST NOT overlap with the primers used in library preparation. *mixed* should only be used if the procedure used for library construction will likely produce multiple categories of sequences in the given experiment. It SHOULD NOT be used as a replacement of a NULL value.
`physical_linkage`	string	required	In case an experimental setup is used that physically links nucleic acids derived from distinct *Rearrangements* before library preparation, this field describes the mode of that linkage. All *hetero_** terms indicate that in case of paired-read sequencing, the two reads should be expected to map to distinct IG/TR loci. **_head-head* refers to techniques that link the 5' ends of transcripts in a single-cell context. **_tail-head* refers to techniques that link the 3' end of one transcript to the 5' end of another one in a single-cell context. This term does not provide any information whether a continuous reading-frame between the two is generated. **_prelinked* refers to constructs in which the linkage was already present on the DNA level (e.g. scFv).

PCR Target Locus Fields

Download as TSV

Name	Type	Attributes	Definition
pcr_target_locus	string	required, nullable	Designation of the target locus. Note that this field uses a controlled vocubulary that is meant to provide a generic classification of the locus, not necessarily the correct designation according to a specific nomenclature.
forward_pcr_primer_target_location	string	required, nullable	Position of the most distal nucleotide templated by the forward primer or primer mix
reverse_pcr_primer_target_location	string	required, nullable	Position of the most proximal nucleotide templated by the reverse primer or primer mix

Raw Sequence Data Fields

Download as TSV

Name	Type	Attributes	Definition
file_type	string	required, nullable	File format for the raw reads or sequences
filename	string	required, nullable	File name for the raw reads or sequences. The first file in paired-read sequencing.
read_direction	string	required, nullable	Read direction for the raw reads or sequences. The first file in paired-read sequencing.
read_length	integer	required, nullable	Read length in bases for the first file in paired-read sequencing
paired_filename	string	required, nullable	File name for the second file in paired-read sequencing
paired_read_direction	string	required, nullable	Read direction for the second file in paired-read sequencing
paired_read_length	integer	required, nullable	Read length in bases for the second file in paired-read sequencing

Sequencing Run Fields

Download as TSV

Name	Type	Attributes	Definition
`sequencing_run_id`	string	required, nullable	ID of sequencing run assigned by the sequencing facility
`total_reads_passing_qc_filter`	integer	required, nullable	Number of usable reads for analysis
`sequencing_platform`	string	required, nullable	Designation of sequencing instrument used
`sequencing_facility`	string	required, nullable	Name and address of sequencing facility
`sequencing_run_date`	string	required, nullable	Date of sequencing run
`sequencing_kit`	string	required, nullable	Name, manufacturer, order and lot numbers of sequencing kit
`sequencing_files`	*RawSequenceData*	optional	Set of sequencing files produced by the sequencing run

Data Processing Fields

`Download as TSV`

Name	Type	Attributes	Definition
`data_processing_id`	string	optional, identifier, nullable	Identifier for the data processing object.
`primary_annotation`	boolean	optional, identifier	If true, indicates this is the primary or default data processing for the repertoire and its rearrangments. If false, indicates this is a secondary or additional data processing.
`software_versions`	string	required, nullable	Version number and / or date, include company pipelines
`paired_reads_assembly`	string	required, nullable	How paired end reads were assembled into a single receptor sequence
`quality_thresholds`	string	required, nullable	How sequences were removed from (4) based on base quality scores
`primer_match_cutoffs`	string	required, nullable	How primers were identified in the sequences, were they removed/masked/etc?
`collapsing_method`	string	required, nullable	The method used for combining multiple sequences from (4) into a single sequence in (5)
`data_processing_protocols`	string	required, nullable	General description of how QC is performed
`data_processing_files`	array of string	optional, nullable	Array of file names for data produced by this data processing.
`germline_database`	string	required, nullable	Source of germline V(D)J genes with version number or date accessed.
`analysis_provenance_id`	string	optional, nullable	Identifier for machine-readable PROV model of analysis provenance

Rearrangement Schema

A Rearrangement is a sequence which describes a rearranged adaptive immune receptor chain (e.g., antibody heavy chain or TCR beta chain) along with a host of annotations. These annotations are defined by the AIRR Rearrangement schema and comprises eight categories.

Category	Description
Input	The input sequence to the V(D)J assignment process.
Identifiers	Primary and foreign key identifiers for linking AIRR data across files and databases.
Primary Annotations	The primary outputs of the V(D)J assignment process, which includes the gene locus, V, D, J, and C gene calls, various flags, V(D)J junction sequence, copy number (`duplicate_count`), and the number of reads contributing to a consensus input sequence (`consensus_count`).
Alignment Annotations	Detailed alignment annotations including the input and germline sequences used in the alignment; score, identity, statistical support (E-value, likelihood, etc); and the alignment itself through CIGAR strings for each aligned gene.
Alignment Positions	The start/end positions for genes in both the input and germline sequences.
Region Sequence	Sequence annotations for the framework regions (FWRs) and complementarity-determining regions (CDRs).
Region Positions	Positional annotations for the framework regions (FWRs) and complementarity-determining regions (CDRs).
Junction Lengths	Lengths for junction sub-regions associated with aspects of the V(D)J recombination process.

File Format Specification

Data for `Rearrangement` or `Alignment` objects are stored as rows in a *tab-delimited* file and should be compatible with any TSV reader. A dataset is defined in this context as: a TSV file, a TSV with a companion YAML file containing metadata, or a directory containing multiple TSV files and YAML files.

Encoding

- The file should be encoded as ASCII or UTF-8.
- Everything is case-sensitive.

Dialect

- The record separator is a newline `\n` and the field separator is a tab `\t`.
- Fields or data should not be quoted.
- A header line with the AIRR-specified column names is always required.
- Values must not contain tab or newline characters.
- Values should avoid `@`, `#`, and quote (`"` or `'`) characters, as the result may be implementation dependent.
- Nested delimiters are not supported by the schema explicitly and should be avoided. However, if multiple values must be reported in a single column for an application specific reason, then the use of a comma as the delimiter is recommended.

File names

AIRR formatted TSV files should end with `.tsv`.

File Structure

The data file has two sections in this order:

1. Header. A single line with column names.
2. Data values. One record per line.

A comment section preceding the header (e.g., `#` or `@` blocks) is not part of the specification, but such a section is reserved for potential inclusion in a future release. As such, a comment section should not be included in the file as it *may* be incompatible with a future specification.

Header

A single line containing the column names and specifying the field order. Any field that corresponds to one of the defined fields should use the specified field name.

Required columns

Some of the fields are defined as `required` and therefore must always be present in the header. Note, however, that all columns allow for null values. Therefore, required columns exist to define a core set of fields that are always present in the table structure, but do not mandate that a value be reported.

Custom columns

There are no restrictions on inclusion of additional custom columns in the Rearrangements file, provided such columns do not use the same name as an existing required or optional field. It is recommended that custom fields follow the same naming scheme as existing fields. Meaning, `snake_case` with narrowing scope when read from left to right. For example, `sequence_id` is the "*identifier* of the *query sequence*".

Consider submitting a pull request for a field name reservation to the airr-standards repository if the field may be broadly useful.

Ordering

There are no requirements that fields or records be sorted or ordered in any specific way. However, the field ordering provided by the schema is a recommended default, with top-to-bottom equating to left-to-right.

Data Values

The possible data types are `string`, `boolean`, `number` (floating point), `integer`, and `null` (empty string).

Boolean values

Boolean values must be encoded as `T` for true and `F` for false.

Null values

All fields may contain null values. This includes columns that are described as `required`. A null value should be encoded as an empty string.

Coordinate numbering

All alignment sequence coordinates use the same scheme as IMGT and INSDC (DDBJ, ENA, GenBank), with the exception that partial coordinate information should not be used in favor of simply assigning the start/end of the alignment. Meaning, coordinates should be provided as 1-based values with closed intervals, without the use of > or < annotations that denoted a partial region.

CIGAR specification

Alignments details are specified using the CIGAR format as defined in the SAM specifications, with some vocabulary restrictions on the use of clipping, skipping, and padding operators.

The CIGAR string defines the reference sequence as the germline sequence of the given gene or region; e.g., for `v_cigar` the reference is the V gene germline sequence. The query sequence is what was input into the alignment tool, which must correspond to what is contained in the `sequence` field of the Rearrangement data. For the majority of use cases, this will necessarily exclude alignment spacers from the CIGAR string, such as IMGT numbering gaps. However, any gaps appearing in the query sequence should be accounted for in the CIGAR string so that the alignment between the query and reference is correctly represented.

The valid operator sets and definitions are as follows:

Operator	Description
=	An identical non-gap character.
X	A differing non-gap character.
M	A positional match in the alignment. This can be either an identical (=) or differing (x) non-gap character.
D	Deletion in the query (gap in the query).
I	Insertion in the query (gap in the reference).
S	Positions that appear in the query, but not the reference. Used exclusively to denote the start position of the alignment in the query. Should precede any N operators.
N	A space in the alignment. Used exclusively to denote the start position of the alignment in the reference. Should follow any S operators.

Note, the use of either the `=`/`X` or `M` syntax is valid, but should be used consistently. While leading `S` and `N` operators are required, tailing `S` and `N` operators are optional.

For example, an D gene alignment that starts at position 419 in the query `sequence` (leading `418S`), that is 16 nucleotides long with no indels (middle `16M`), has an 10 nucleotide 5’ deletion (leading `10N`), a 5 nucleotide 3’ deletion (trailing `5N`), and ends 72 nucleotides from the end of the query `sequence` (trailing `71S`) would have the following D gene CIGAR string (`d_cigar`) and positional information:

Field	Value
d_cigar	418S10N16M71S5N
d_sequence_start	419
d_sequence_end	434
d_germline_start	11
d_germline_end	26

Definition Clarifications

Junction versus CDR3

We work with the IMGT definitions of the junction and CDR3 regions. Specifically, the IMGT `JUNCTION` includes the conserved cysteine and tryptophan/phenylalanine residues, while `CDR3` excludes those two residues. Therefore, our `junction` and `junction_aa` fields which represent the extracted sequence include the two conserved residues, while the coordinate fields (`cdr3_start` and `cdr3_end`) exclude them.

Productive

The schema does not define a strict definition of a productive rearrangement. However, the IMGT definition is recommended:

1. Coding region has an open reading frame
2. No defect in the start codon, splicing sites or regulatory elements.
3. No internal stop codons.
4. An in-frame junction region.

Locus names

A naming convention for locus names is not strictly enforced, but the IMGT locus names are recommended. For example, in the case of human data, this would be the set: IGH, IGK, IGL, TRA, TRB, TRD, or TRG.

Gene and allele names

Gene call examples use the IMGT nomenclature, but no specific gene or allele nomenclature is strictly mandated. Species denotations may or may not be included in the gene name, as appropriate. For example, “Homo sapiens IGHV4-59*01”, “IGHV4-59*01” and “AB019438” are all valid entries for the same allele.

However, when using an established reference database to assign gene calls adherence to the exact nomenclature used by the reference database is strongly recommended, as this will facilitate mapping to the database entries, cross-study comparison, and upload to public repositories.

Alignments

There is no required alignment scheme for the nucleotide and amino acid alignment fields. These fields may, or may not, include numbering spacers (e.g., IMGT-numbering gaps), variations in case to denote mismatches, deletions, or other features appropriate to the tool that performed the alignment. The only strict requirement is that the query (“sequence”) and reference (“germline”) **must** be properly aligned.

Fields

The specification includes two classes of fields. Those that are required and those that are optional. Required is defined as a column that must be present in the header of the TSV. Optional is defined as column that may, or may not, appear in the TSV. All fields, including required fields, are nullable by assigning an empty string as the value. There are no requirements for column ordering in the schema, although the Python and R reference APIs enforce ordering for the sake of generating predictable output. The set of optional fields that provide alignment and region coordinates (“_start” and "_end" fields) are defined as 1- based closed intervals, similar to the SAM, VCF, GFF, IMGT, and INDSC formats (GenBank, ENA, and DDJB; http://www.insdc.org).

Most fields have strict definitions for the values that they contain. However, some commonly provided information cannot be standardized across diverse toolchains, so a small selection of fields have context-dependent definitions. In particular, these context-dependent fields include the optional "_score," "_identity," and "_support" fields used for assessing the quality of alignments which vary considerably in definition based on the methodology used. Similarly, the "_alignment" fields require strict alignment between the corresponding observed and germline sequences, but the manner in which that alignment is conveyed is somewhat flexible in that it allows for any numbering scheme (e.g., IMGT or KABAT) or lack thereof.

By default, data elements representing sequences in the schema contain nucleotide sequences except for data elements ending in "_aa," which are amino acid translations of the associated nucleotide sequence.

While the format contains an extensive list of reserved field names, there are no restrictions on inclusion of custom fields in the TSV file, provided such custom fields have a unique name. Furthermore, suggestions for extending the format with additional reserved names are welcomed through the issue tracker on the GitHub repository (https://github.com/airr-community/airr-standards).

`Download as TSV`

Name	Type	Attributes	Definition
`sequence_id`	string	required, identifier, nullable	Unique query sequence identifier for the Rearrangment. Most often this will be the input sequence header or a substring thereof, but may also be a custom identifier defined by the tool in cases where query sequences have been combined in some fashion prior to alignment. When downloaded from an AIRR Data Commons repository, this will usually be a universally unique record locator for linking with other objects in the AIRR Data Model.
`sequence`	string	required, nullable	The query nucleotide sequence. Usually, this is the unmodified input sequence, which may be reverse complemented if necessary. In some cases, this field may contain consensus sequences or other types of collapsed input sequences if these steps are performed prior to alignment.
`sequence_aa`	string	optional, nullable	Amino acid translation of the query nucleotide sequence.
`rev_comp`	boolean	required, nullable	True if the alignment is on the opposite strand (reverse complemented) with respect to the query sequence. If True then all output data, such as alignment coordinates and sequences, are based on the reverse complement of 'sequence'.
`productive`	boolean	required, nullable	True if the V(D)J sequence is predicted to be productive.
`vj_in_frame`	boolean	optional, nullable	True if the V and J gene alignments are in-frame.
`stop_codon`	boolean	optional, nullable	True if the aligned sequence contains a stop codon.

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
`complete_vdj`	boolean	optional, nullable	True if the sequence alignment spans the entire V(D)J region. Meaning, sequence_alignment includes both the first V gene codon that encodes the mature polypeptide chain (i.e., after the leader sequence) and the last complete codon of the J gene (i.e., before the J-C splice site). This does not require an absence of deletions within the internal FWR and CDR regions of the alignment.
`locus`	string	optional, nullable	Gene locus (chain type). Note that this field uses a controlled vocabulary that is meant to provide a generic classification of the locus, not necessarily the correct designation according to a specific nomenclature.
`v_call`	string	required, nullable	V gene with allele. If referring to a known reference sequence in a database the relevant gene/allele nomenclature should be followed (e.g., IGHV4-59*01 if using IMGT/GENE-DB).
`d_call`	string	required, nullable	First or only D gene with allele. If referring to a known reference sequence in a database the relevant gene/allele nomenclature should be followed (e.g., IGHD3-10*01 if using IMGT/GENE-DB).
`d2_call`	string	optional, nullable	Second D gene with allele. If referring to a known reference sequence in a database the relevant gene/allele nomenclature should be followed (e.g., IGHD3-10*01 if using IMGT/GENE-DB).
`j_call`	string	required, nullable	J gene with allele. If referring to a known reference sequence in a database the relevant gene/allele nomenclature should be followed (e.g., IGHJ4*02 if using IMGT/GENE-DB).
`c_call`	string	optional, nullable	Constant region gene with allele. If referring to a known reference sequence in a database the relevant gene/allele nomenclature should be followed (e.g., IGHG1*01 if using IMGT/GENE-DB).
`sequence_alignment`	string	required, nullable	Aligned portion of query sequence, including any indel corrections or numbering spacers, such as IMGT-gaps. Typically, this will include only the V(D)J region, but that is not a requirement.
`sequence_alignment_aa`	string	optional, nullable	Amino acid translation of the aligned query sequence.
`germline_alignment`	string	required, nullable	Assembled, aligned, full-length inferred germline sequence spanning the same region as the sequence_alignment field (typically the V(D)J region) and including the same set of corrections and spacers (if any).
`germline_alignment_aa`	string	optional, nullable	Amino acid translation of the assembled germline sequence.
`junction`	string	required, nullable	Junction region nucleotide sequence, where the junction is defined as the CDR3 plus the two flanking conserved codons.
`junction_aa`	string	required, nullable	Amino acid translation of the junction.

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
np1	string	optional, nullable	Nucleotide sequence of the combined N/P region between the V gene and first D gene alignment or between the V gene and J gene alignments.
np1_aa	string	optional, nullable	Amino acid translation of the np1 field.
np2	string	optional, nullable	Nucleotide sequence of the combined N/P region between either the first D gene and J gene alignments or the first D gene and second D gene alignments.
np2_aa	string	optional, nullable	Amino acid translation of the np2 field.
np3	string	optional, nullable	Nucleotide sequence of the combined N/P region between the second D gene and J gene alignments.
np3_aa	string	optional, nullable	Amino acid translation of the np3 field.
cdr1	string	optional, nullable	Nucleotide sequence of the aligned CDR1 region.
cdr1_aa	string	optional, nullable	Amino acid translation of the cdr1 field.
cdr2	string	optional, nullable	Nucleotide sequence of the aligned CDR2 region.
cdr2_aa	string	optional, nullable	Amino acid translation of the cdr2 field.
cdr3	string	optional, nullable	Nucleotide sequence of the aligned CDR3 region.
cdr3_aa	string	optional, nullable	Amino acid translation of the cdr3 field.
fwr1	string	optional, nullable	Nucleotide sequence of the aligned FWR1 region.
fwr1_aa	string	optional, nullable	Amino acid translation of the fwr1 field.
fwr2	string	optional, nullable	Nucleotide sequence of the aligned FWR2 region.
fwr2_aa	string	optional, nullable	Amino acid translation of the fwr2 field.
fwr3	string	optional, nullable	Nucleotide sequence of the aligned FWR3 region.
fwr3_aa	string	optional, nullable	Amino acid translation of the fwr3 field.
fwr4	string	optional, nullable	Nucleotide sequence of the aligned FWR4 region.
fwr4_aa	string	optional, nullable	Amino acid translation of the fwr4 field.
v_score	number	optional, nullable	Alignment score for the V gene.
v_identity	number	optional, nullable	Fractional identity for the V gene alignment.
v_support	number	optional, nullable	V gene alignment E-value, p-value, likelihood, probability or other similar measure of support for the V gene assignment as defined by the alignment tool.

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
v_cigar	string	required, nullable	CIGAR string for the V gene alignment.
d_score	number	optional, nullable	Alignment score for the first or only D gene alignment.
d_identity	number	optional, nullable	Fractional identity for the first or only D gene alignment.
d_support	number	optional, nullable	D gene alignment E-value, p-value, likelihood, probability or other similar measure of support for the first or only D gene as defined by the alignment tool.
d_cigar	string	required, nullable	CIGAR string for the first or only D gene alignment.
d2_score	number	optional, nullable	Alignment score for the second D gene alignment.
d2_identity	number	optional, nullable	Fractional identity for the second D gene alignment.
d2_support	number	optional, nullable	D gene alignment E-value, p-value, likelihood, probability or other similar measure of support for the second D gene as defined by the alignment tool.
d2_cigar	string	optional, nullable	CIGAR string for the second D gene alignment.
j_score	number	optional, nullable	Alignment score for the J gene alignment.
j_identity	number	optional, nullable	Fractional identity for the J gene alignment.
j_support	number	optional, nullable	J gene alignment E-value, p-value, likelihood, probability or other similar measure of support for the J gene assignment as defined by the alignment tool.
j_cigar	string	required, nullable	CIGAR string for the J gene alignment.
c_score	number	optional, nullable	Alignment score for the C gene alignment.
c_identity	number	optional, nullable	Fractional identity for the C gene alignment.
c_support	number	optional, nullable	C gene alignment E-value, p-value, likelihood, probability or other similar measure of support for the C gene assignment as defined by the alignment tool.
c_cigar	string	optional, nullable	CIGAR string for the C gene alignment.
v_sequence_start	integer	optional, nullable	Start position of the V gene in the query sequence (1-based closed interval).
v_sequence_end	integer	optional, nullable	End position of the V gene in the query sequence (1-based closed interval).
v_germline_start	integer	optional, nullable	Alignment start position in the V gene reference sequence (1-based closed interval).
v_germline_end	integer	optional, nullable	Alignment end position in the V gene reference sequence (1-based closed interval).
v_alignment_start	integer	optional, nullable	Start position of the V gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
v_alignment_end	integer	optional, nullable	End position of the V gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d_sequence_start	integer	optional, nullable	Start position of the first or only D gene in the query sequence. (1-based closed interval).
d_sequence_end	integer	optional, nullable	End position of the first or only D gene in the query sequence. (1-based closed interval).
d_germline_start	integer	optional, nullable	Alignment start position in the D gene reference sequence for the first or only D gene (1-based closed interval).
d_germline_end	integer	optional, nullable	Alignment end position in the D gene reference sequence for the first or only D gene (1-based closed interval).
d_alignment_start	integer	optional, nullable	Start position of the first or only D gene in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d_alignment_end	integer	optional, nullable	End position of the first or only D gene in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d2_sequence_start	integer	optional, nullable	Start position of the second D gene in the query sequence (1-based closed interval).
d2_sequence_end	integer	optional, nullable	End position of the second D gene in the query sequence (1-based closed interval).
d2_germline_start	integer	optional, nullable	Alignment start position in the second D gene reference sequence (1-based closed interval).
d2_germline_end	integer	optional, nullable	Alignment end position in the second D gene reference sequence (1-based closed interval).
d2_alignment_start	integer	optional, nullable	Start position of the second D gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d2_alignment_end	integer	optional, nullable	End position of the second D gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
j_sequence_start	integer	optional, nullable	Start position of the J gene in the query sequence (1-based closed interval).
j_sequence_end	integer	optional, nullable	End position of the J gene in the query sequence (1-based closed interval).
j_germline_start	integer	optional, nullable	Alignment start position in the J gene reference sequence (1-based closed interval).
j_germline_end	integer	optional, nullable	Alignment end position in the J gene reference sequence (1-based closed interval).
j_alignment_start	integer	optional, nullable	Start position of the J gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
j_alignment_end	integer	optional, nullable	End position of the J gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
cdr1_start	integer	optional, nullable	CDR1 start position in the query sequence (1-based closed interval).

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
cdr1_end	integer	optional, nullable	CDR1 end position in the query sequence (1-based closed interval).
cdr2_start	integer	optional, nullable	CDR2 start position in the query sequence (1-based closed interval).
cdr2_end	integer	optional, nullable	CDR2 end position in the query sequence (1-based closed interval).
cdr3_start	integer	optional, nullable	CDR3 start position in the query sequence (1-based closed interval).
cdr3_end	integer	optional, nullable	CDR3 end position in the query sequence (1-based closed interval).
fwr1_start	integer	optional, nullable	FWR1 start position in the query sequence (1-based closed interval).
fwr1_end	integer	optional, nullable	FWR1 end position in the query sequence (1-based closed interval).
fwr2_start	integer	optional, nullable	FWR2 start position in the query sequence (1-based closed interval).
fwr2_end	integer	optional, nullable	FWR2 end position in the query sequence (1-based closed interval).
fwr3_start	integer	optional, nullable	FWR3 start position in the query sequence (1-based closed interval).
fwr3_end	integer	optional, nullable	FWR3 end position in the query sequence (1-based closed interval).
fwr4_start	integer	optional, nullable	FWR4 start position in the query sequence (1-based closed interval).
fwr4_end	integer	optional, nullable	FWR4 end position in the query sequence (1-based closed interval).
v_sequence_alignment	string	optional, nullable	Aligned portion of query sequence assigned to the V gene, including any indel corrections or numbering spacers.
v_sequence_alignment_aa	string	optional, nullable	Amino acid translation of the v_sequence_alignment field.
d_sequence_alignment	string	optional, nullable	Aligned portion of query sequence assigned to the first or only D gene, including any indel corrections or numbering spacers.
d_sequence_alignment_aa	string	optional, nullable	Amino acid translation of the d_sequence_alignment field.
d2_sequence_alignment	string	optional, nullable	Aligned portion of query sequence assigned to the second D gene, including any indel corrections or numbering spacers.
d2_sequence_alignment_aa	string	optional, nullable	Amino acid translation of the d2_sequence_alignment field.
j_sequence_alignment	string	optional, nullable	Aligned portion of query sequence assigned to the J gene, including any indel corrections or numbering spacers.
j_sequence_alignment_aa	string	optional, nullable	Amino acid translation of the j_sequence_alignment field.
c_sequence_alignment	string	optional, nullable	Aligned portion of query sequence assigned to the constant region, including any indel corrections or numbering spacers.

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
c_sequence_alignment_aa	string	optional, nullable	Amino acid translation of the c_sequence_alignment field.
v_germline_alignment	string	optional, nullable	Aligned V gene germline sequence spanning the same region as the v_sequence_alignment field and including the same set of corrections and spacers (if any).
v_germline_alignment_aa	string	optional, nullable	Amino acid translation of the v_germline_alignment field.
d_germline_alignment	string	optional, nullable	Aligned D gene germline sequence spanning the same region as the d_sequence_alignment field and including the same set of corrections and spacers (if any).
d_germline_alignment_aa	string	optional, nullable	Amino acid translation of the d_germline_alignment field.
d2_germline_alignment	string	optional, nullable	Aligned D gene germline sequence spanning the same region as the d2_sequence_alignment field and including the same set of corrections and spacers (if any).
d2_germline_alignment_aa	string	optional, nullable	Amino acid translation of the d2_germline_alignment field.
j_germline_alignment	string	optional, nullable	Aligned J gene germline sequence spanning the same region as the j_sequence_alignment field and including the same set of corrections and spacers (if any).
j_germline_alignment_aa	string	optional, nullable	Amino acid translation of the j_germline_alignment field.
c_germline_alignment	string	optional, nullable	Aligned constant region germline sequence spanning the same region as the c_sequence_alignment field and including the same set of corrections and spacers (if any).
c_germline_alignment_aa	string	optional, nullable	Amino acid translation of the c_germline_aligment field.
junction_length	integer	optional, nullable	Number of nucleotides in the junction sequence.
junction_aa_length	integer	optional, nullable	Number of amino acids in the junction sequence.
np1_length	integer	optional, nullable	Number of nucleotides between the V gene and first D gene alignments or between the V gene and J gene alignments.
np2_length	integer	optional, nullable	Number of nucleotides between either the first D gene and J gene alignments or the first D gene and second D gene alignments.
np3_length	integer	optional, nullable	Number of nucleotides between the second D gene and J gene alignments.
n1_length	integer	optional, nullable	Number of untemplated nucleotides 5’ of the first or only D gene alignment.
n2_length	integer	optional, nullable	Number of untemplated nucleotides 3’ of the first or only D gene alignment.
n3_length	integer	optional, nullable	Number of untemplated nucleotides 3’ of the second D gene alignment.
p3v_length	integer	optional, nullable	Number of palindromic nucleotides 3’ of the V gene alignment.
p5d_length	integer	optional, nullable	Number of palindromic nucleotides 5’ of the first or only D gene alignment.

Continued on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
`p3d_length`	integer	optional, nullable	Number of palindromic nucleotides 3' of the first or only D gene alignment.
`p5d2_length`	integer	optional, nullable	Number of palindromic nucleotides 5' of the second D gene alignment.
`p3d2_length`	integer	optional, nullable	Number of palindromic nucleotides 3' of the second D gene alignment.
`p5j_length`	integer	optional, nullable	Number of palindromic nucleotides 5' of the J gene alignment.
`consensus_count`	integer	optional, nullable	Number of reads contributing to the (UMI) consensus for this sequence. For example, the sum of the number of reads for all UMIs that contribute to the query sequence.
`duplicate_count`	integer	optional, nullable	Copy number or number of duplicate observations for the query sequence. For example, the number of UMIs sharing an identical sequence or the number of identical observations of this sequence absent UMIs.
`cell_id`	string	optional, identifier, nullable	Identifier defining the cell of origin for the query sequence.
`clone_id`	string	optional, identifier, nullable	Clonal cluster assignment for the query sequence.
`repertoire_id`	string	optional, identifier, nullable	Identifier to the associated repertoire in study metadata.
`sample_processing_id`	string	optional, identifier, nullable	Identifier to the sample processing object in the repertoire metadata for this rearrangement. If the repertoire has a single sample then this field may be empty or missing. If the repertoire has multiple samples then this field may be empty or missing if the sample cannot be differentiated or the relationship is not maintained by the data processing.
`data_processing_id`	string	optional, identifier, nullable	Identifier to the data processing object in the repertoire metadata for this rearrangement. If this field is empty than the primary data processing object is assumed.
`rearrangement_id`	string	DEPRECATED	Identifier for the Rearrangement object. May be identical to sequence_id, but will usually be a univerally unique record locator for database applications.
`rearrangement_set_id`	string	DEPRECATED	Identifier for grouping Rearrangement objects.
`germline_database`	string	DEPRECATED	Source of germline V(D)J genes with version number or date accessed.

Alignment Schema (Experimental)

An Alignment is the output from a V(D)J assignment process for a single V, D, J, or C gene for a sequence. It is not necessary that the V(D)J assignment process performs a sequence alignment algorithm, as the schema can support any algorithmic process. Multiple Alignment records are supported and expected for a single sequence with context-dependent fields (`score`, `identity`, `support`, `rank`) for assessing the quality of assignments that can vary considerably in definition based on the methodology used.

Note, this schema definition is still experimental and should not be considered final.

File Format Specification

The *format specification* describes the file format and details on how to structure this data.

Fields

`Download as TSV`

Name	Type	Attributes	Definition
`sequence_id`	string	required, nullable	Unique query sequence identifier within the file. Most often this will be the input sequence header or a substring thereof, but may also be a custom identifier defined by the tool in cases where query sequences have been combined in some fashion prior to alignment.
`segment`	string	required, nullable	The segment for this alignment. One of V, D, J or C.
`rev_comp`	boolean	optional, nullable	Alignment result is from the reverse complement of the query sequence.
`call`	string	required, nullable	Gene assignment with allele.
`score`	number	required, nullable	Alignment score.
`identity`	number	optional, nullable	Alignment fractional identity.
`support`	number	optional, nullable	Alignment E-value, p-value, likelihood, probability or other similar measure of support for the gene assignment as defined by the alignment tool.
`cigar`	string	required, nullable	Alignment CIGAR string.
`sequence_start`	integer	optional, nullable	Start position of the segment in the query sequence (1-based closed interval).
`sequence_end`	integer	optional, nullable	End position of the segment in the query sequence (1-based closed interval).
`germline_start`	integer	optional, nullable	Alignment start position in the reference sequence (1-based closed interval).
`germline_end`	integer	optional, nullable	Alignment end position in the reference sequence (1-based closed interval).
`rank`	integer	optional, nullable	Alignment rank.
`rearrangement_id`	string	optional, nullable	Identifier for the Rearrangement object. May be identical to sequence_id, but will usually be a universally unique record locator for database applications.
`data_processing_id`	string	optional, nullable	Identifier to the data processing object in the repertoire metadata for this rearrangement. If this field is empty than the primary data processing object is assumed.
`germline_database`	string	DEPRECATED	Source of germline V(D)J genes with version number or date accessed.

Clone and Lineage Tree Schema (Experimental)

A unique inferred clone object that has been constructed within a single data processing for a single repertoire and a subset of its sequences and/or rearrangements.

A clone object may have one or more inferred lineage trees. Each tree is represented by a Newick string for its edges and a dictionary of node objects.

File Format Specification

The file format has not been specified yet.

Clone Fields

`Download as TSV`

Name	Type	Attributes	Definition
clone_id	string	required, nullable	Identifier for the clone.
repertoire_id	string	optional, nullable	Identifier to the associated repertoire in study metadata.
data_processing_id	string	optional, nullable	Identifier of the data processing object in the repertoire metadata for this clone.
sequences	array of string	optional, nullable	List sequence_id strings that act as keys to the Rearrangement records for members of the clone.
v_call	string	optional, nullable	V gene with allele of the inferred ancestral of the clone. For example, IGHV4-59*01.
d_call	string	optional, nullable	D gene with allele of the inferred ancestor of the clone. For example, IGHD3-10*01.
j_call	string	optional, nullable	J gene with allele of the inferred ancestor of the clone. For example, IGHJ4*02.
junction	string	optional, nullable	Nucleotide sequence for the junction region of the inferred ancestor of the clone, where the junction is defined as the CDR3 plus the two flanking conserved codons.
junction_aa	string	optional, nullable	Amino acid translation of the junction.
junction_length	integer	optional, nullable	Number of nucleotides in the junction.
junction_aa_length	integer	optional, nullable	Number of amino acids in junction_aa.
germline_alignment	string	required, nullable	Assembled, aligned, full-length inferred ancestor of the clone spanning the same region as the sequence_alignment field of nodes (typically the V(D)J region) and including the same set of corrections and spacers (if any).
germline_alignment_aa	string	optional, nullable	Amino acid translation of germline_alignment.
v_alignment_start	integer	optional, nullable	Start position in the V gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
v_alignment_end	integer	optional, nullable	End position in the V gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d_alignment_start	integer	optional, nullable	Start position of the D gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
d_alignment_end	integer	optional, nullable	End position of the D gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
j_alignment_start	integer	optional, nullable	Start position of the J gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
j_alignment_end	integer	optional, nullable	End position of the J gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
junction_start	integer	optional, nullable	Junction region start position in the alignment (1-based closed interval).
junction_end	integer	optional, nullable	Junction region end position in the alignment (1-based closed interval).
sequence_count	integer	optional, nullable	Number of Rearrangement records (sequences) included in this clone
seed_id	string	optional, nullable	sequence_id of the seed sequence. Empty string (or null) if there is no seed sequence.

Tree Fields

Download as TSV

Name	Type	Attributes	Definition
tree_id	string	required, nullable	Identifier for the tree.
clone_id	string	required, nullable	Identifier for the clone.
newick	string	required, nullable	Newick string of the tree edges.
nodes	object	optional, nullable	Dictionary of nodes in the tree, keyed by sequence_id string

Node Fields

Download as TSV

Name	Type	Attributes	Definition
sequence_id	string	required, nullable	Identifier for this node that matches the identifier in the newick string and, where possible, the sequence_id in the source repertoire.
sequence_alignment	string	optional, nullable	Nucleotide sequence of the node, aligned to the germline_alignment for this clone, including including any indel corrections or spacers.
junction	string	optional, nullable	Junction region nucleotide sequence for the node, where the junction is defined as the CDR3 plus the two flanking conserved codons.
junction_aa	string	optional, nullable	Amino acid translation of the junction.

Cell Schema (Experimental)

The cell object acts as point of reference for all data that can be related to an individual cell, either by direct observation or inference.

File Format Specification

The file format has not been specified yet.

Cell Fields

Download as TSV

Name	Type	Attributes	Definition
`cell_id`	string	required	Identifier defining the cell of origin for the query sequence.
`rearrangements`	array of string	required, nullable	Array of sequence identifiers defined for the Rearrangement object
`receptors`	array of string	optional, nullable	Array of receptor identifiers defined for the Receptor object
`repertoire_id`	string	required, nullable	Identifier to the associated repertoire in study metadata.
`data_processing_id`	string	optional, nullable	Identifier of the data processing object in the repertoire metadata for this clone.
`expression_study_method`	string	optional, nullable	keyword describing the methodology used to assess expression. This values for this field MUST come from a controlled vocabulary
`expression_raw_doi`	string	optional, nullable	DOI of raw data set containing the current event
`expression_index`	string	optional, nullable	Index addressing the current event within the raw data set.
`expression_tabular`	array of object	optional, nullable	Expression definitions for single-cell
`virtual_pairing`	boolean	required, nullable	boolean to indicate if pairing was inferred.

2.3.3 AIRR Software WG - Guidance for AIRR Software Tools

Version 1.0

AIRR Software WG - Compliance Checklist for AIRR Software Tools

Version 1.0 (when finalised)

This questionnaire should be read in conjunction with the *AIRR Software WG - Guidance for AIRR Software Tools*.

To submit your tool for ratification against the standard, please send the completed questionnaire to software@airrc.antibodysociety.org.

Please provide comments in italics in each response box where these would be helpful to facilitate understanding. We kindly ask for a brief explanatory comment if your answer to a question is *no* or *not applicable*.

Name of Tool:

Contact Name/Institution:

Contact email:

Requirement Ref.	Question	Response
1	Where is the source code published (please provide a link)?	
2	Does the tool support *AIRR Data Representations* standards? Please list any other standard data formats that are supported	yes/no
3	Does the distribution include example data? Is the example data in MiAIRR format, where applicable? Does the tool provide automated checks for expected output from example data?	yes/no yes/no/not applicable yes/no
4	Does the output of the tool include a summary of the run parameters?	yes/no
5	Is a container build file provided? Container technology used? Is the container automatically built as new versions are released? Does the automated build run the tool against the example data and test the output?	yes/no Docker/Singularity/Other (please specify) yes/no yes/no
6	Where can users see what level of support is available? (Please provide a link)	
7	Under what software licence is the tool published? (please provide the name of the licence (e.g. GPL, MIT) or a link	

AIRR Software WG - List of Tools Certified as Compliant

The following tools have been certified as compliant with v1.0 of the guidelines:

Software	Version	Support	Reference
SONAR	3	Output	Schramm et al. Front Immunol, 2016.

Evaluation Data Sets

The Software WG is working on the development and evaluation of simulated data sets.

Lists of published real-world datasets are maintained in the AIRR Forum Wiki.

Introduction

The Adaptive Immune Receptor Repertoire (AIRR) Community will benefit greatly from cooperation among groups developing software tools and resources for AIRR research. The goal of the AIRR Software Working Group is to promote standards for AIRR software tools and resources in order to enable rigorous and reproducible immune repertoire research at the largest scale possible. As one contribution to this goal, we have established the following standards for software tools. Authors whose tools comply with this standard will, subject to ratification from the AIRR Software WG, be permitted to advertise their tools as being AIRR-compliant.

Requirements

Tools must:

1. Be published in source code form, and hosted on a publicly available repository with a clear versioning system.

2. Support community-curated standard file formats and strive for modularity and interoperability with other tools. In particular, tools must read and write *AIRR Data Representations* standards corresponding to their tool.
3. Include example data (in AIRR standard formats where applicable) and an automated check for expected output from that data, in order to provide a minimal example of functionality allowing users to check that the software is performing as described.
4. Provide information about run parameters as part of the output.
5. Provide a container build file that can be used to create an image which encapsulates the software tool, its dependencies, and required run environment. This needs to be remotely and automatically built. The build should conclude by running the example data through the tool (see point 3) and confirming that the expected output is obtained. We currently recognize two software solutions, although we will adapt as software evolves:
    a. A Dockerfile that automatically builds a container image on Docker Hub.
    b. A Singularity recipe file that automatically builds a container image on Singularity Hub.
6. Provide user support, clearly stating which level of support users can expect, and how and from whom to obtain it.

Recommendations

We suggest software tools be published under a license that permits free access, use, modification, and sharing, such as GPL, Apache 2.0, or MIT. However, we understand that this depends on institutional intellectual property restrictions, thus it is a recommendation rather than a requirement.

Explanatory Notes

Open Source Software and Versioned Repositories

Software tools in the AIRR field are evolving rapidly. In the interests of reproducibility and transparency, published work should be based on tools (and versions of tools) that can be obtained easily by other researchers in the future. To that end, AIRR compliant tools must be published in open repositories such as GitHub or Bitbucket, and we encourage publishing users to provide specifics on the version and configuration of tools that have been employed.

Community-Curated File Formats

The AIRR Data Representation Working Group has defined standards for immune receptor repertoire sequencing datasets. Software tool authors are requested to support these standards as much as possible, for applicable data sets. The currently implemented standard covers submission of reads to NCBI repositories (BioProject, BioSample, SRA and Genbank) and annotated immune receptor rearrangements. Tool authors can assist by easing/guiding the process of submission as much as possible.

Example Data and Checks

Because the installation and operation of the tools in this field may be complex, we require example data and details of expected output, so that users can confirm that their installation is functioning as expected. Furthermore, metadata (for example, germline gene libraries) and other software dependencies should be checked when the tool runs, and informative error messages issued if necessary. A means should be provided to check the expected output automatically.

Dependencies and Containers

Containers encapsulate everything needed to run a piece of software into a single convenient executable that is largely independent of the user's software environment. For the following purposes, providers of AIRR-compliant tools must provide a containerized implementation (based on a published build script as described above) as one download option that users can choose:

- Containers allow users to use and evaluate a tool easily and reproduce results, without the need to resolve dependencies or configure the environment.
- Having these containers be automatically built also provides a self-validated way to understand the fine details of installation from a known starting point.

To ensure that containers are up to date, they must be built automatically when the current release version of the tool is updated. We will use automated builds on Docker Hub and Singularity Hub for this purpose. The corresponding build files document dependencies clearly, and make it easy for the maintainer to keep the container's dependencies up to date in subsequent releases.

An example Docker container is provided on the Software WG Github Repository. This example encapsulates IgBLAST, and implements the bioboxes command-line standard.

Support Statements

Tool authors must provide support for the tool. They must state explicitly what level of support is provided, and explain how support can be obtained. We recommend a method such as the issues tracker on Github, that publishes support requests transparently and links resolutions to specific versions or releases. Users are advised to check that the level of support and the frequency of software updates matches their expectations before committing to a tool.

Analysis Workflows

- At the moment, we do not endorse a specific workflow technology standard:
  - Technology is evolving too rapidly for us to commit to a particular workflow.
  - Typically, AIRR analysis tools have many options and modes, which would make it difficult to support a "plug and play" environment without unduly restricting functionality.
- As tools and workflows evolve, we will keep the position under review and may make stronger technology recommendations in the future.
- We strongly encourage authors of tools to provide concrete, documented, examples of workflows that employ their tools, together with sample input and output data.
- **Likewise we encourage authors of research publications to provide** documented workflows that will enable interested readers to reproduce the results.

Ratification

Authors may submit tools to the AIRR Software WG requesting ratification against the standard. The submitter should provide a completed copy of the *AIRR Software WG - Compliance Checklist for AIRR Software Tools* to evidence reviewable and itemised evidence of compliance with each Requirement listed above.

The Software WG will, where appropriate, issue a Certificate of Compliance, stating the version of the tool reviewed and the version of the Standard with which compliance was ratified. After receiving a Certificate, authors will be entitled to claim compliance with the Standard, and may incorporate any artwork provided by AIRR for that purpose.

The Software WG will maintain and publish a list of compliant software.

If a tool does not achieve ratification, the Software WG will provide an explanation. The Software WG encourages resubmission once issues have been resolved.

Authors must re-submit tools for ratification following major upgrades or substantial modifications. The Software WG may, at its discretion, request resubmission at any time. If a certified tool subsequently fails ratification, or is not re-submitted in response to a request from the Software WG, AIRR compliance may no longer be claimed and the associated artwork may no longer be used.

The Software WG may, at its discretion, issue a new version of this standard at any time. Tools certified against previous version(s) of the standard may continue to claim compliance with those versions and to use the associated artwork. Authors wishing to claim compliance with the new version must submit a new request for certification and may not claim compliance with the new version, or use associated artwork, until and unless certification is obtained.

2.3.4 AIRR Data Commons API V1

The use of high-throughput sequencing for profiling B-cell and T-cell receptors has resulted in a rapid increase in data generation. It is timely, therefore, for the Adaptive Immune Receptor Repertoire (AIRR) community to establish a clear set of community-accepted data and metadata standards; analytical tools; and policies and practices for infrastructure to support data deposit, curation, storage, and use. Such actions are in accordance with international funder and journal policies that promote data deposition and data sharing – at a minimum, data on which scientific publications are based should be made available immediately on publication. Data deposit in publicly accessible databases ensures that published results may be validated. Such deposition also facilitates reuse of data for the generation of new hypotheses and new knowledge.

The AIRR Common Repository Working Group (CRWG) developed a set of recommendations (v0.6.0) that promote the deposit, sharing, and use of AIRR sequence data. These recommendations were refined following community discussions at the AIRR 2016 and 2017 Community Meetings and were approved through a vote by the AIRR Community at the AIRR Community Meeting in December 2017.

Overview

The AIRR Data Commons (ADC) API provides programmatic access to query and download AIRR-seq data. The ADC API uses JSON as its communication format, and standard HTTP methods like `GET` and `POST`. The ADC API is read-only and the mechanism of inclusion of AIRR-seq studies into a data repository is left up to the repository.

This documentation explains how to construct and execute API requests and interpret API responses.

API Endpoints

The ADC API is versioned with the version number (`v1`) as part of the base path for all endpoints. Each ADC API endpoint represents specific functionality as summarized in the following table:

Endpoint	Type	HTTP	Description
/v1	Service status	GET	Returns success if API service is running.
/v1/info	Service information	GET	Upon success, returns service information such as name, version, etc.
/v1/repertoire/{repertoire_id}	Retrieve a repertoire given its repertoire_id	GET	Upon success, returns the Repertoire information in JSON according to the *Repertoire schema*.
/v1/repertoire	Query repertoires	POST	Upon success, returns a list of Repertoires in JSON according to the *Repertoire schema*.
/v1/rearrangement/{sequence_id}	Retrieve a rearrangement given its sequence_id	GET	Upon success, returns the Rearrangement information in JSON format according to the *Rearrangement schema*.
/v1/rearrangement	Query rearrangements	POST	Upon success, returns a list of Rearrangements in JSON or AIRR TSV format according to the *Rearrangement schema*.

Authentication

The ADC API currently does not define an authentication method. Future versions of the API will provide an authentication method so data repositories can support query and download of controlled-access data.

Search and Retrieval

The AIRR Data Commons API specifies endpoints for searching and retrieving AIRR-seq data sets stored in an AIRR-compliant Data Repository according to the AIRR Data Model. This documentation describes Version 1 of the API. The general format of requests and associated parameters are described below.

The design of the AIRR Data Commons API was greatly inspired by National Cancer Institute's Genomic Data Commons (GDC) API.

Components of a Request

The ADC API has two classes of endpoints. The endpoints that respond to GET requests are simple services that require few or no parameters. While, the endpoints that response to POST requests are the main query services and provide many parameters for specifying the query as well as the data in the API response.

A typical POST query request specifies the following parameters:

- The filters parameter specifies the query.
- The from and size parameters specify the number of results to skip and the maximum number of results to be returned in the response.
- The fields parameter specifies which data elements to be returned in the response. By default all fields (AIRR and non-AIRR) stored in the data repository are returned. This can vary between data repositories based upon how the repository decides to store blank or null fields, so the fields and/or include_fields parameter should be used to guarantee the existence of data elements in the response.
- The include_fields parameter specifies the set of AIRR fields to be included in the response. This parameter can be used in conjunction with the fields parameter, in which case the list of fields is merged. This is a mechanism to ensure that specific, well-defined sets of AIRR data elements are returned without requiring all of those fields to be individually provided in the fields parameter.

The sets that can be requested are summarized in the table below.

include_fields	MiAIRR	AIRR required	AIRR identifiers	other AIRR fields
miairr	Y	some	N	N
airr-core	Y	Y	Y	N
airr-schema	Y	Y	Y	Y

Service Status Example

The following is an example `GET` request to check that the service API is available for VDJServer's data repository.

```
curl https://vdjserver.org/airr/v1
```

The response should indicate success.

```
{"result":"success"}
```

Service Info Example

The following is an example `GET` request to get information about the service.

```
curl https://vdjserver.org/airr/v1
```

The response provides various information.

```
{
  "name": "adc-api-js-mongodb",
  "description": "AIRR Data Commons API reference implementation",
  "version": "1.0.0",
  "airr_schema_version": 1.3,
  "max_size": 1000,
  "max_query_size": 2097152,
  "contact": {
    "name": "AIRR Community",
    "url": "https://github.com/airr-community"
  }
}
```

Query Repertoire Example

The following is an example `POST` request to the `repertoire` endpoint of the ADC API. It queries for repertoires of human TCR beta receptors (`filters`), skips the first 10 results (`from`), requests 5 results (`size`), and requests only the `repertoire_id` field (`fields`).

```
curl --data @query1-2_repertoire.json https://vdjserver.org/airr/v1/repertoire
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
                "content": {
                    "field":"subject.organism.id",
                    "value":"9606"
                }
            },
```

(continues on next page)

(continued from previous page)

```
            {
                "op":"=",
                "content": {
                    "field":"sample.pcr_target.pcr_target_locus",
                    "value":"TRB"
                }
            }
        ]
    },
    "from":10,
    "size":5,
    "fields":["repertoire_id"]
}
```

The response contains two JSON objects, an Info object that provides information about the API response and a Repertoire object that contains the list of Repertoires that met the query search criteria. In this case, the query returns a list of five repertoire identifiers. Note the Info object is based on the info block as specified in the OpenAPI v2.0 specification.

```
{
  "Info":
  {
      "title": "AIRR Data Commons API reference implementation",
      "description": "API response for repertoire query",
      "version": 1.3,
      "contact":
      {
          "name": "AIRR Community",
          "url": "https://github.com/airr-community"
      }
  },
  "Repertoire":
  [
      {"repertoire_id": "4357957907784536551-242ac11c-0001-012"},
      {"repertoire_id": "4476756703191896551-242ac11c-0001-012"},
      {"repertoire_id": "6205695788196696551-242ac11c-0001-012"},
      {"repertoire_id": "6393557657723736551-242ac11c-0001-012"},
      {"repertoire_id": "7158276584776536551-242ac11c-0001-012"}
  ]
}
```

Endpoints

The ADC API V1 provides two primary endpoints for querying and retrieving AIRR-seq data. The `repertoire` endpoint allows querying upon any field in the *Repertoire schema* including study, subject, sample, cell processing, nucleic acid processing, sequencing run, raw sequencing files, and data processing information. Queries on the content of raw sequencing files is not support but is supported on file attributes such as name, type and read information. Queries on `Rearrangements` is provided by the `rearrangement` endpoint.

The standard workflow to retrieve all of the data for an AIRR-seq study involves performing a query on the `repertoire` endpoint to retrieve the repertoires in the study, and one or more queries on the `rearrangement` endpoint to download the rearrangement data for each repertoire. The endpoints are designed so the API response can be saved directly into a file and be used by AIRR analysis tools, including the AIRR python and R reference libraries, without requiring modifications or transformation of the data.

Repertoire Endpoint

The repertoire endpoint provides access to all fields in the *Repertoire schema*. There are two type of endpoints; one for retrieving a single repertoire given its identifier, and another for performing a query across all repertoires in the data repository.

It is expected that the number of repertoires in a data repository will never become so large such that queries become computationally expensive. A data repository might have thousands of repertoires across hundreds of studies, yet such numbers are easily handled by modern databases. Based upon this, the ADC API does not place limits on the repertoire endpoint for the fields that can be queried, the operators that can be used, or the number of results that can be returned.

Retrieve a Single Repertoire

Given a repertoire_id, a single Repertoire object will be returned.

```
curl https://vdjserver.org/airr/v1/repertoire/4357957907784536551-242ac11c-0001-012
```

The response will provide the Repertoire data in JSON format.

```
{
  "Info":
  {
      "title": "AIRR Data Commons API reference implementation",
      "description": "API response for repertoire query",
      "version": 1.3,
      "contact":
      {
          "name": "AIRR Community",
          "url": "https://github.com/airr-community"
      }
  },
  "Repertoire":
  [
    {
      "repertoire_id":"4357957907784536551-242ac11c-0001-012",
      "study":{
         "study_id":"PRJNA300878",
         "submitted_by":"Florian Rubelt",
         "pub_ids":"PMID:27005435",
         "lab_name":"Mark M. Davis",
         "lab_address":"Stanford University",
         "study_title":"Homo sapiens B and T cell repertoire - MZ twins"
      },
      "subject":{
         "subject_id":"TW02A",
         "synthetic":false,
         "linked_subjects":"TW02B",
         "organism":{"id":"9606","value":"Homo sapiens"},
         "age":"25yr",
         "link_type":"twin",
         "sex":"F"
      },
      "sample":[
        {"sample_id":"TW02A_T_memory_CD4",
         "pcr_target":[{"pcr_target_locus":"TRB"}],
         "cell_isolation":"FACS",
         "read_length":"300",
         "cell_phenotype":"expression of CD45RO and CCR7",
```

(continues on next page)

(continued from previous page)

```
         "cell_subset":"Memory CD4+ T cell",
         "filename":"SRR2905669_R1.fastq.gz",
         "single_cell":false,
         "file_type":"fastq",
         "tissue":"PBMC",
         "template_class":"RNA",
         "paired_filename":"SRR2905669_R2.fastq.gz",
         "paired_read_direction":"reverse",
         "read_direction":"forward",
         "sequencing_platform":"Illumina MiSeq"}
      ],
      "data_processing":[
        {"data_processing_id":"4976322832749171176-242ac11c-0001-012",
         "analysis_provenance_id":"651223970338378216-242ac11b-0001-007"}
      ]
    }
  ]
}
```

Query against all Repertoires

A query in JSON format is passed in a `POST` request. This example queries for repertoires of human IG heavy chain receptors for all studies in the data repository.

```
curl --data @query2_repertoire.json https://vdjserver.org/airr/v1/repertoire
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
                "content": {
                    "field":"subject.organism.id",
                    "value":"9606"
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"sample.pcr_target.pcr_target_locus",
                    "value":"IGH"
                }
            }
        ]
    }
}
```

The response will provide a list of `Repertoires` in JSON format. The example output is not provided here due to its size.

Rearrangement Endpoint

The `rearrangement` endpoint provides access to all fields in the *Rearrangement schema*. There are two type of endpoints; one for retrieving a single rearrangement given its identifier, and another for performing a query across all rearrangements in the data repository.

Unlike repertoire data, data repositories are expected to store millions or billions of rearrangement records, where performing "simple" queries can quickly become computationally expensive. Data repositories will need to optimize their databases for performance. Therefore, the ADC API does not require that all fields be queryable and only a limited set of query capabilities must be supported. The queryable fields are described in the Fields section below.

Retrieve a Single Rearrangement

Given a `sequence_id`, a single `Rearrangement` object will be returned.

```
curl https://vdjserver.org/airr/v1/rearrangement/5d6fba725dca5569326aa104
```

The response will provide the `Rearrangement` data in JSON format.

```
{
  "Info":
  {
      "title": "AIRR Data Commons API reference implementation",
      "description": "API response for rearrangement query",
      "version": 1.3,
      "contact":
      {
          "name": "AIRR Community",
          "url": "https://github.com/airr-community"
      }
  },
  "Rearrangement":
  [
    {
      "sequence_id":"5d6fba725dca5569326aa104",
      "repertoire_id":"1841923116114776551-242ac11c-0001-012",

      "... remaining fields":"snipped for space"
    }
  ]
}
```

Query against all Rearrangements

Supplying a `repertoire_id`, when it is known, should greatly speed up the query as it can significantly reduce the amount of data to be searched, though it isn't necessary.

This example queries for rearrangements with a specific junction amino acid sequence among a set of repertoires. A limited set of fields is requested to be returned. The resultant data can be requested in JSON or *AIRR TSV* format.

```
curl --data @query1_rearrangement.json https://vdjserver.org/airr/v1/rearrangement
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":[
                        "2366080924918616551-242ac11c-0001-012",
                        "2541616238306136551-242ac11c-0001-012",
```

(continues on next page)

(continued from previous page)

```
                        "1993707260355416551-242ac11c-0001-012",
                        "1841923116114776551-242ac11c-0001-012"
                    ]
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
            }
        ]
    },
    "fields":["repertoire_id","sequence_id","v_call","productive"],
    "format":"tsv"
}
```

Here is the response in AIRR TSV format.

```
productive    v_call  sequence_id     repertoire_id
true  IGHV1-69*04     5d6fba725dca5569326aa106        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba725dca5569326aa11b        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*10     5d6fba725dca5569326aa149        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba735dca5569326aa245        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba735dca5569326aa274        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba735dca5569326aa27b        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba735dca5569326aa27c        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-24*01     5d6fba735dca5569326aa2a0        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba745dca5569326aa359        1841923116114776551-242ac11c-
↪0001-012
true  IGHV1-69*04     5d6fba745dca5569326aa408        1841923116114776551-242ac11c-
↪0001-012
```

Request Parameters

The ADC API supports the follow query parameters. These are only applicable to the `repertoire` and `rearrangement` query endpoints, i.e. the HTTP `POST` endpoints.

Parameter	Default	Description
`filters`	null	Specifies logical expression for query critieria
`format`	JSON	Specifies the API response format: JSON, AIRR TSV
`include_fields`	null	Specifies the set of AIRR fields to be included in the response
`fields`	null	Specifies which fields to include in the response
`from`	0	Specifies the first record to return from a set of search results
`size`	repository dependent	Specifies the number of results to return
`facets`	null	Provide aggregate count information for the specified fields

Filters Query Parameter

The `filters` parameter enables passing complex query criteria to the ADC API. The parameter represents the query in a JSON object.

A `filters` query consists of an operator (or a nested set of operators) with a set of `field` and `value` operands. The query criteria as represented in a JSON object can be considered an expression tree data structure where internal nodes are operators and child nodes are operands. The expression tree can be of any depth, and recursive algorithms are typically used for tree traversal.

The following operators are support by the ADC API.

Operator	Operands	Value Data Types	Description	Example
=	field and value	string, number, integer, or boolean	equals	{“op”:”=”,”content”:{“field”:”junction_aa”,”value”:”CASSYIKLN”}}
!=	field and value	string, number, integer, or boolean	does not equal	{“op”:”!=”,”content”:{“field”:”subject.organism.id”,”value”:”9606”}}
<	field and value	number, integer	less than	{“op”:”<”,”content”:{“field”:”sample.cell_number”,”value”:1000}}
<=	field and value	number, integer	less than or equal	{“op”:”<=”,”content”:{“field”:”sample.cell_number”,”value”:1000}}
>	field and value	number, integer	greater than	{“op”:”>”,”content”:{“field”:”sample.cells_per_reaction”,”value”:10000}}
>=	field and value	number, integer	greater than or equal	{“op”:”>=”,”content”:{“field”:”sample.cells_per_reaction”,”value”:10000}}
is missing	field	n/a	field is missing or is null	{“op”:”is missing”,”content”:{“field”:”sample.tissue”}}
is	field	n/a	identical to “is missing” operator, provided for GDC compatibility	{“op”:”is”,”content”:{“field”:”sample.tissue”}}
is not missing	field	n/a	field is not missing and is not null	{“op”:”is not missing”,”content”:{“field”:”sample.tissue”}}
not	field	n/a	identical to “is not missing” operator, provided for GDC compatibility	{“op”:”not”,”content”:{“field”:”sample.tissue”}}
in	field, multiple values in a list	array of string, number, or integer	matches a string or number in a list	{“op”:”in”,”content”:{“field”:”subject.strain_name”,”value”:[“C57BL/6”,”BALB/c”,
exclude	field, multiple values in a list	array of string, number, or integer	does not match any string or number in a list	{“op”:”exclude”,”content”:{“field”:”subject.strain_name”,”value”:[“SCID”,”NOD”]
contains	field, value	string	contains the substring	{“op”:”contains”,”content”:{“field”:”study.study_title”,”value”:”cancer”}}
and	multiple	n/a	logical AND	{“op”:”and”,”content”:[{“op”:”!=”,”content”:{“field”:”subject.organism.id”,”value” {“op”:”>=”,”content”:{“field”:”sample.cells_per_reaction”,”value”:10000}},

Note that the not operator is different from a logical NOT operator, and the logical NOT is not needed as the other operators provide negation.

The field operand specifies a fully qualified property name in the AIRR Data Model. Fully qualified AIRR properties are either a JSON/YAML base type (string, number, integer, or boolean) or an array of one of these base types (some AIRR fields are arrays e.g. study.keywords_study). The Fields section below describes the available queryable fields.

The value operand specifies one or more values when evaluating the operator for the field operand.

Queries Against Arrays

A number of fields in the AIRR Data Model are arrays, such as study.keywords_study which is an array of strings or subject.diagnosis which is an array of Diagnosis objects. A query operator on an array field will apply that operator to each entry in the array to decide if the query filter is satisfied. The behavior is different for various operators. For operators such as = and in, the filter behaves like the Boolean OR over the array entries, that is if **any** array entry evaluates to true then the query filter is satisfied. For operators such as != and exclude, the filter behaves like the Boolean AND over the array entries, that is **all** array entries must evaluate to true for the query filter to be satisfied.

Examples

A simple query with a single operator looks like this:

```
{
  "filters": {
    "op":"=",
    "content": {
      "field":"junction_aa",
      "value":"CASSYIKLN"
    }
  }
}
```

A more complex query with multiple operators looks like this:

```
{
  "filters": {
    "op":"and",
    "content": [
      {
        "op":"!=",
        "content": {
          "field":"subject.organism.id",
          "value":"9606"
        }
      },
      {
        "op":">=",
        "content": {
          "field":"sample.cells_per_reaction",
          "value":"10000"
        }
      },
      {
        "op":"exclude",
        "content": {
          "field":"subject.organism.id",
          "value": ["9606", "10090"]
```

(continues on next page)

(continued from previous page)

```
        }
      }
    ]
  }
}
```

Format Query Parameter

Specifies the format of the API response. `json` is the default format and is available for all endpoints. The `rearrangement` `POST` endpoint also accepts `tsv` which will provide the data in the *AIRR TSV* format.

Fields Query Parameter

The `fields` parameter specifies which fields are to be included in the API response. By default all fields (AIRR and non-AIRR) stored in the data repository are returned. However, this can vary between data repositories based upon how the repository decides to store blank or null fields, so the `fields` and/or `include_fields` parameter should be used to guarantee the existence of data elements in the response.

Include Fields Query Parameter

The `include_fields` parameter specifies that the API response should include a well-defined set of AIRR Standard fields. These sets include:

- `miairr`, for only the MiAIRR fields.
- `airr-core`, for the AIRR required and identifier fields. This is expected to be the most common option as it provides all MiAIRR fields, additional required fields useful for analysis, and all identifier fields for linking objects in the AIRR Data Model.
- `airr-schema`, for all AIRR fields in the AIRR Schema.

The `include_fields` parameter is a mechanism to ensure that specific AIRR data elements are returned without requiring those fields to be individually provided with the `fields` parameter. Any data elements that lack a value will be assigned `null` in the response. Any empty array of objects, for example `subject.diagnosis`, will be populated with a single object with all of the object's properties given a null value. Any empty array of primitive data types, like string or number, will be assigned `null`. Note that if both the `include_fields` and the `fields` parameter are provided, the API response will include the set of AIRR fields and in addition will include any additional fields that are specified in the `fields` parameter.

Size and From Query Parameters

The ADC API provides a pagination feature that limits the number of results returned by the API.

The `from` query parameter specifies which record to start from when returning results. This allows records to be skipped. The default value is `0` indicating that the first record in the set of results will be returned.

The `size` query parameters specifies the maximum number of results to return. The default value is specific to the data repository, and a maximum value may be imposed by the data repository. This is to prevent queries from "accidently" returning millions of records. The `info` endpoint provides the data repository default and maximum values for the `repertoire` and `rearrangement` endpoints, which may have different values. A value of `0` indicates there is no limit on the number of results to return, but if the data repository does not support this then the default value will be used.

The combination of `from` and `size` can be used to implement pagination in a graphical user interface, or to split a very large download into smaller batches. For example, if an interface displays 10 records as a time, the request would assign `size=10` and `from=0` to get the ten results to display on the first page. When the user traverses to the "next page", the request would assign `from=10` to skip the first ten results and return the next ten results, and `from=20` for the next page after that, and so on.

Facets Query Parameter

The `facets` parameter provides aggregate count information for the specified field. Only a single field can be specified. The `facets` parameter can be used in conjunction with the `filters` parameter to get aggregate counts for a set of search results. It returns the set of values for the field, and the number of records (repertoires or rearrangement) that have this value. For field values that have no counts, the API service can either return the field value with a 0 count or exclude the field value in the aggregation. The typical use of this parameter is for displaying aggregate information in a graphical user interface.

Here is a simple query with only the `facets` parameter to return the set of values for `sample.pcr_target.pcr_target_locus` and the count of repertoires repertoires that have each value. The content of the `JSON payload`.

```
{
    "facets":"sample.pcr_target.pcr_target_locus"
}
```

Sending this query in an API request.

```
curl --data @facets1_repertoire.json https://vdjserver.org/airr/v1/repertoire
```

The output from the request is similar to normal queries except the data is provided with the *Facet* key.

```
{
  "Info": {
    "title": "AIRR Data Commons API reference implementation",
    "description": "API response for repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-community"
    }
  },
  "Facet": [
    {"sample.pcr_target.pcr_target_locus":[["TRB"]],"count":40},
    {"sample.pcr_target.pcr_target_locus":[["IGH"]],"count":20}
  ]
}
```

Here is a query with both `filters` and `facets` parameters, which restricts the data records used for the facets count. The content of the `JSON payload`.

```
{
    "filters":{
        "op":"=",
        "content": {
            "field":"sample.pcr_target.pcr_target_locus",
            "value":"IGH"
        }
    },
    "facets":"subject.subject_id"
}
```

Sending this query in an API request.

```
curl --data @facets2_repertoire.json https://vdjserver.org/airr/v1/repertoire
```

Example output from the request. This result indicates there are ten subjects each with two IGH repertoires.

```
{
  "Info": {
    "title": "AIRR Data Commons API reference implementation",
    "description": "API response for repertoire query",
    "version": 1.3,
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-community"
    }
  },
  "Facet": [
    {"subject.subject_id":"TW05B","count":2},
    {"subject.subject_id":"TW05A","count":2},
    {"subject.subject_id":"TW03A","count":2},
    {"subject.subject_id":"TW04A","count":2},
    {"subject.subject_id":"TW01A","count":2},
    {"subject.subject_id":"TW04B","count":2},
    {"subject.subject_id":"TW02A","count":2},
    {"subject.subject_id":"TW03B","count":2},
    {"subject.subject_id":"TW01B","count":2},
    {"subject.subject_id":"TW02B","count":2}
  ]
}
```

ADC API Limits and Thresholds

Repertoire endpoint query fields

It is expected that the number of repertoires in a data repository will never become so large such that queries become computationally expensive. A data repository might have thousands of repertoires across hundreds of studies, yet such numbers are easily handled by databases. Based upon this, the ADC API does not place limits on the repertoire endpoint for the fields that can be queried or the operators that can be used.

Rearrangement endpoint query fields

Unlike repertoire data, data repositories are expected to store billions of rearrangement records, where performing "simple" queries can quickly become computationally expensive. Data repositories are encouraged to optimize their databases for performance. Therefore, based upon a set of query use cases provided by immunology experts, a minimal set of required fields was defined that can be queried. These required fields are described in the following Table. The fields also have the AIRR extension property `adc-query-support: true` in the AIRR Schema.

Field(s)	Description
sequence_id, repertoire_id, sample_processing_id, data_processing_id, clone_id, cell_id	Identifiers; sequence_id allows for query of that specific rearrangement object in the repository, while repertoire_id, sample_processing_id, and data_processing_id are links to the repertoire metadata for the rearrangement. The clone_id and cell_id are identifiers that group rearrangements based on clone assignment and single cell assignment.
locus, v_call, d_call, j_call, c_call, productive, junction_aa, junction_aa_length	Commonly used rearrangement annotations.

Repertoire/rearrangement object size

Any single repertoire or rearrangement object has a maximum that is typically dependent upon the back-end database which stores the data. For MongoDB-based data repositories, the largest object size is 16 megabytes.

Repertoire/rearrangement query size

For MongoDB-based data repositories, a query is a document thus the query size is limited to the maximum document size of 16 megabytes.

Data repository specific limits

A data repository may provide additional limits. These can be retrieved from the `info` endpoint. If the data repository does not provide a limit, then the ADC API default limit or no limit is assumed.

Field	Description
`max_size`	The maximum value for the `size` query parameter. Attempting to retrieve beyond this maximum may trigger an error or may only return `max_size` records based upon the data repository behavior.
`max_query_size`	The maximum size of the JSON query object.

Reference Implementation

The AIRR Community provides a reference implementation for an ADC API service with more information found *here*.

2.3.5 AIRR Ontologies and Vocabularies Team

Summary

The "Ontologies and Vocabularies Team" was formed as a joint interest group of the Common Repository (ComRepo) and the Minimal Standards (MiniStd) working groups of the AIRR Community. The long-term aim of the Team is to define standard vocabularies and ontologies to be used by AIRR-compliant databases.

Sprint Reports

OntoVoc Report - Sprint 11/2018

Objectives

The objectives of this first sprint in November 2018 were to:

1. define criteria for suitable ontologies
2. identify ontologies for five fields/keywords of the MiAIRR data standard and
3. assess technical aspects of ontology integration into databases

General Considerations

The Team initially discussed an approach where only vocabularies (i.e. lists of terms) and not ontologies (i.e. many terms connected by predicates) would have been defined. These vocabularies would have been derived from ontologies, but this process would not necessarily have been reversible. The notion at this time point was, that such an approach would allow to solve a number of problems like combining multiple sources and removing duplicated leaves. However, after some discussions this approach was effectively abandoned for a number of reasons:

- It would discard the UID for an entity. As the UID (in contrast to the name string) is guaranteed to be stable and unique, it facilitates updates, linking and information representation, all of which would otherwise be lost.

- In general, it will be more sustainable to work with the maintainers of an existing ontology to include entities/terms, than just dumping their terms into a list and adding new ones.
- Well-designed ontologies will not contain duplicated entities, although they might appear to do so in a simple browsers (i.e. this is an artifact of representation). Ontologies that actually do contain duplicates are excluded by *criterium 2*.

Criteria for Ontologies

Criteria

Ontologies used within AIRR standards

1. MUST[1] cover the majority of the required terms, but complete coverage is OPTIONAL
2. MUST have a structure that is scientifically correct and logically coherent
3. MUST NOT feature complexity that makes it hard to use for queries and data representation
4. SHOULD already be widely adopted
5. MUST be actively maintained
6. MUST be available under a free license

Comments on criteria:

- ad *(1)*: For most fields it will be difficult to find complete and accurate ontologies. Therefore picking the best available ontology and working with its maintainers to include missing terms is expected to be the most sustainable approach.
- ad *(5)*: This requirement follows from *(1)*, as there needs to be a way for term requests.
- ad *(6)*: A number of ontologies need to be licensed from their respective copyright holders. This results in potential barriers for implementation and distribution of such ontologies. Therefore only ontologies available under a free license are considered suitable for AIRR-compliant databases. The list of suitable licenses is not final, but includes: CC0 and CC BY.

Selected Ontologies

(designations are MiAIRR field names and `DataRep keywords`)

Completed

- Species (`organism`)
  - NCBITAXON
  - license: UMLS[2]
  - latest release: 2018-07-06
  - maintainer: NCBI (info@ncbi.nlm.nih.gov)
- Diagnosis (`disease_diagnosis`)

[1] See the "Glossary" section on how to interpret term written in all-caps.

[2] Will require further review the UMLS Metathesaurus License is not a free license, however it needs to be clarified how much of it relates to the work (i.e. the taxonomy itself) and how much to the service.

  - DOID
  - root node
    - name: `disease`
    - ID: `DOID:4`
    - path: `/disease`
  - License: CC BY
  - latest release: 2018-03-02
  - maintainer: Lynn Schriml, U Maryland, MD, US (lynn.schriml@gmail.com)
  - notes: Features ICD cross-reference
- Cell subset (`cell_subset`)
  - CellOntology
  - license: CC BY
  - latest release: 2018-07-11
  - maintainer: Alexander Diehl, Buffalo, NY, US (addiehl@buffalo.edu)
- Tissue (`tissue`)
  - Uberon
  - root node
    - name: `multicellular anatomical structure`
    - ID: `UBERON:0010000`
    - path: `/BFO_0000002/BFO_0000004/anatomical entity/material anatomical entity/anatomical structure/multicellular anatomical structure`
  - License: CC BY
  - latest release: 2018-10-15
  - Maintainer: Chris Mungall, LBL, CA, US (cjmungall@lbl.gov)

Under evaluation

- Strain name (`strain_name`)
  - Suggested ontologies:
    - JAX
    - IEDB
  - Issues:
    - Nomenclature
    - one ontology is not enough

Technical aspects

- Repositories:
  - UID assigned by ontologies are guaranteed to be unique and permanent[3].
  - A repository MAY use internal identifiers that are distinct from UIDs. However, to be AIRR-compliant it MUST be able to map UIDs to its identifiers.
  - Points of "AIRR compliance" would typically be:
    - When data is extracted from the repository through a Query API (CRWG)
    - When data is extracted from the repository into a file format (DataRep)
- Integration of ontologies into repositories:
  - There are two main ontology providers offering a REST API and all the ontologies listed above:
    - NCBO Bioportal
    - OLS ontology
  - NCBO can apparently be slow and sometimes not that stable, while OLS seems to be more stable and potentially has a better long-term support.
  - Remote ontology services tend to be slow and create external dependencies. On the other hand, while local hosting of an ontology is possible (and partially supported by NCBO and OLS), it requires non-negligible resources. The Team's current assumption is that queries to remote ontology services can be substantially accelerated if only the relevant section of a respective ontology is queried. Therefore a local service would not be necessary.
  - Repositories should store both the IDs and the values in their database. This way, they do not have to query the ontology in a scenario where human-readable output is required. In the case of changes, most ontologies try to follow the practice of not changing a term value but instead create a new term with the new value and a new ID, and deprecating the old term. Therefore term deprecation needs to be handled by the repository.
  - Like for the databases, also the API should be able to handle both IDs and values as query input and return both during a query.
  - The user interface (UI) should offer an ontology-backed autocomplete. NCBO provides some JavaScript code to use. The UI must not offer deprecated terms. To allow entry of terms not present in the ontology, data can be prefixed with some text that will allow the data validation to proceed (e.g., if an entry starts with "other -" the UI will not autocomplete/validate). Later, i.e. when the term has been created, the data will be updated.
- Note that the complete IEDB can be downloaded as SQL dump, it is licensed under CC BY. At a first glance, the main overlap seems to be with `organism`, `strain_name` and to a smaller extent `disease_diagnosis`. However, sample information like `cell_subset` and `tissue` seems to be largely absent from IEDB, so it could currently not be the one-stop solution for AIRR.

Footnotes

Appendix

[3] This has more recently (early 2020) been called in question and will be revisited during the next sprint. Note that the uncertainty revolves around the question what exactly constitues a UID, rather than the question whether a UID is permanent and unique.

Glossary

- MUST / REQUIRED: Indicates that an element or action is necessary to conform to the standard.
- SHOULD / RECOMMENDED: Indicates that an element or action is considered to be best practice by AIRR, but not necessary to conform to the standard.
- MAY / OPTIONAL: Indicates that it is at the discretion of the user to use an element or perform an action.
- MUST NOT / FORBIDDEN: Indicates that an element or action will be in conflict with the standard.

OntoVoc Report - Sprint 04/2020

Objectives

The objectives of this second sprint in April 2020 were to:

1. revisit general policies around ontologies used in the AIRR schema
2. identify two new ontologies for several fields of the AIRR schema
3. solve technical questions regarding IDs and providers

General Policies

The OntoVoc team revisited the criteria for ontologies used in the AIRR schema that it *defined in the 11/2018 sprint*. While they are still considered to be valid, the team felt that a more detailed guidance could be useful in the process of selecting ontologies for new fields. It therefore evaluated the OBO Foundry Principles, which partially re-iterate some of the existing criteria (e.g., *Openness* and *Maintenance*), but also provide additional recommendations, e.g., the presence of textual definitions, clear scope and a common format, which were considered to be valuable additions to the existing guidelines. The team therefore decided to endorse the OBO Foundry Principles, as RECOMMENDED (but NOT REQUIRED) criteria. It should be noted, that this does not make any statement regarding the use of OBO vs. non-OBO ontologies.

Decisions on Pending Items of Sprint 11/2018

A number of decisions on draft and legacy ontologies as well as root nodes was not officially passed during the last sprint. The team thus revisited and confirmed the following decisions:

- Use of NCIT for study_type, top node Study (NCIT:C63536).
- Use of UO for age_unit, top node time unit (UO:0000003).
- Use of `Gnathostomata (NCBITAXON:7776) as top node for NCBITAXON when used for fields encoding a host species.
- Use of lymphocyte (CL:0000542) as top node for CL when used for cell_subset.

New Ontologies

Mouse strain

Background

Mouse strain names follow a very elaborate nomenclature that is capable of describing the genetic background, breeding history and introduced mutation in a detailed manner. However, this nomenclature is rarely used correctly (if at all), which creates uncertainty about the identity of strains used in experimental studies. Therefore an ontology or vocabulary compliant to this nomenclature would be of tremendous help for consistent annotation.

An ontology for the `strain_name` field was already on the list for the last sprint, however it was not possible to identify a single ontology that would contain comprehensive information about strains from multiple species. This situation created a problem that could not be resolved then. In the meantime, the concept of "extensions" has been introduced to the AIRR schema, which create an additional layer of fields (and associated ontologies) on top of a core schema. As these extensions can be made conditional on the value of fields within the core schema, it has now become possible to have multiple extensions defining the `strain_name` field, but for different species and therefore with distinct species-specific ontologies.

Having addressed this issue, the other key problem that remains is the absence of an actual ontology for mouse strains, while a rat strain ontology exists. Therefore in a first step it is necessary to identify resources that you at least serve as a provider for vocabularies. The two potential candidates that were identified are:

- MGI: The Mouse Genome Informatics database hosted at JAX aims to be comprehensive in regard to all mouse strains that have been published in the literature.
- IEDB: The Immune Epitope Database already ran into the problem of a missing mouse ontology and therefore decided to build up their own reference focused on immunologically relevant strains, as part of their Ontie database.

Once it is clear which of the resources could be used, it will be necessary to approach the current maintainers regarding their willingness to convert the data into an actual ontology (the RS could serve as a template for this). As this will take longer than just a couple of weeks, the second step is out-of-scope for this sprint.

Evaluation

- MGI: The database can be downloaded as a dump, however the licensing conditions are unclear. It contains a total of 60k entries of which 3.2k inbred and 13.8k are congenic strains. The majority of the remaining entries are coisogenic strains, most of them from large- scale gene KO projects.
- IEDB: Database dumps can also be downloaded and are freely available under CC-BY 4.0. It covers over a thousand mouse strains and contains additional information on the genetic background of a strain.

Next steps

- Get in touch with JAX (pending)

Geolocation

There are several (planned) extensions to the AIRR metadata standard that will provide geospatial metadata. Country-level information is typically assumed to be privacy-preserving and easy to operationalize. Therefore, while clearly only capturing some aspects of genetic ancestry, it might serve as a proxy for concepts of "race" and "ethnicity" that are rather ill-defined.

Potential candidate vocabularies/ontologies:

- ISO3166-1 alpha-2: Two-letter code, some ambiguity but well known from ccTLDs.
- ISO3166-1 alpha-3: Three-letter code, less ambiguity than alpha-2.

- UN Stats Division code (currently M49): Numerical code, not human-readable, maps to ISO3166-1 alpha-3.
- Gazetter (GAZ)
  - Contains 2nd (state) and 3rd (county) level information.
  - Not linked to any actual coordinates
  - ISO3166-1 annotation is incomplete and lacks e.g. for Germany and Switzerland.
  - Does not support German Umlauts. Äbsölütely inacceptable, as these are not just diacritical marks (i.e. "Münster" and "Munster" are two different cities).
- HANCESTRO:
  - Seems to be complete, but does not provide ISO3166 codes.
  - Ontology could also be used for other fields relating to genetic ancestry.
  - Links to DBpedia, currently unclear whether it is also populated from there
  - *country* node has pan-240 leaves (surplus seems due to oversea territories), cross-referencing to GAZ (s/a)
- Various pathogen-related repositories:
  - VectorBase (VBGEO): see link and choose "GADM/VBGEO PlaceNames"
  - Viral Pathogen Resource (ViPR):
    - Uses v1.3 of the GSCID/BRC Project and Sample Application Standard.
    - GSCID/BRC Core Sample defines four fields for "Collection Location":
      - "Latitude" (`CS11`) and "Longitude" (`CS12`) in ISO 6709 format
      - "Location" (`CS13`), using GAZ as controlled vocabulary
      - "Country" (`CS14`) as by ISO3166-1 (alpha-2).
  - Influenza Research Database (IRD): Flu-focused version of ViPR, also uses GSCID/BRC Project and Sample Application Standard v1.3.
  - Pathosystems Resource Integration Center (Patric): Focused on bacterial infectious diseases. Uses an "Isolation Country" field in their "Genome" table, format seems to be full text.

Rejected candidates:

- HL7: own ontology deprecated, now recommends ISO 3166-1 alpha-3 set.
- NCIT: Incomplete, only contains pan-90 entities
- SNOMED: Licensing issues
- GADM data: Good quality and resolution, but not an ontology in itself. Also not under a free license, does not allow redistribution or commercial use.

Evaluation

Given the number of options, there is no obvious candidate to pick. Therefore the team decided to define clear use cases and then evaluate each options against them. However, due to time limitation, we did not really get into this, will have to follow up in the next sprint. The use cases so far were:

- Annotate country of birth / of sampling [REQUIRED]
- Encode higher resolution than country level if legally permitted and scientifically meaningful [RECOMMENDED].

- Linking to geo-spatial coordinates [OPTIONAL]

Technical Questions

Background and Problem

Some nomenclature first: The nodes in an ontology graph are typically either *concepts* (e.g., capital) or *instances* thereof (e.g., Paris). These nodes have *local IDs* (often numbers), which are unique within an ontology. They also typically have *labels*, which is the human- readable name of the node. Nodes can have additional *attributes* (e.g., "population count") and are connected to other nodes by *relations* (e.g. "is-a", "superset-of"), which create the edges of the graph.

The complete ontology is usually represented in an XML or OWL file. However, we are looking for a *provider*, i.e. a service that facilitates queries of an ontology via web and/or an API-based interface. Upon querying with a unique ID, is it expected that a *provider* will be able to return the record of a node, which should contain all attributes and relations. Furthermore a *provider* might allow set- and graph-based queries (e.g., is A a complete subset of B; what is the last common ancestor of A and B). Finally a *provider* can offer lookup services, i.e., identify the corresponding *concept* or *instance* in another ontology. Until now we have mainly looked at three providers: Ontobee, OLS and BioPortal. While they all provide similar basic services, it should be noted that some biomedical databases and repositories are, by convention, restricted to use certain *providers*.

As stated above, each node has a *local ID*. To avoid conflicts between the *local IDs* of multiple ontologies, *providers* and ontology collections (e.g., OBO Foundry) use a namespace, i.e., some abbreviation for the ontology that is prefixed to the *local ID*. However, as there no common standard how to create these prefixes, this system is only unambiguous and collision-safe within a single *provider*. To resolve this issue, ontologies often use International Resource Identifiers (IRI, [RFC3987]). While IRIs look like HTTP URLs, they should primarily be considered as permanent and globally unique identifiers, which might resolve to the node's record via DNS/HTTP, but this is optional. In addition, potential intermediate URLs generated in the DNS/HTTP resolving process must be considered internal and therefore should not be used by third parties. Finally, it needs be noted that IRIs should to be considered case-sensitive, especially when used as identifiers (per [RFC3987], Section 5.3.2.1, which only excludes the schema and host (authority) component for case-sensitivity).

While many ontologies already define an entities IRI on the level of the ontology, there are some that do not. For such ontologies, IRIs are then assigned by the provider. The most notable example for this are the UMLS ontologies like the NCBI Taxonomy. This leads to the situation that a single node in an ontology, stored by two providers can have different IRIs. Therefore, a concept from NCBI Taxonomy, e.g., the duck-billed platypus (`label`: *Ornithorhynchus anatinus*, local ID: 9258) has the IRI `http://purl.obolibrary.org/obo/NCBITaxon_9258` in Ontobee and the IRI `http://purl.bioontology.org/ontology/NCBITAXON/9258` in BioPortal. In addition, other providers might choose to use one of these IRIs too, although it will never resolve to their system via DNS/HTTP (e.g., OLS uses the Ontobee IRIs).

For the AIRR Community, this creates the challenge that we want to be able to have unambiguous identifiers, without requiring any specific provider.

Proposed solution

Compact URIs (CURIEs) are a standardized way to abbreviate IRIs, which includes URIs as a subset. They were originally conceived to simplify the handling of attributes, e.g. in XML or SPARQL, by making them more compact and readable. CURIEs are e.g. used by IEDB databases to reduce redundancies (mainly in the leading part of IRIs).

A typical CURIE would, e.g., look like `NCBITAXON:9258`. In this case, `NCBITAXON` is the *prefix*, a custom string that will be replaced by a repository-defined IRI component (e.g., `http://purl.obolibrary.org/obo/NCBITaxon_`). Note that there is no connection between `NCBITAXON` in the CURIE and `NCBITaxon` in the IRI, the former one is just a placeholder.

This resolves the issue of different *providers* usings different IRIs with distinct formatting rules (as described above). As the choice of the *provider* is independent for each ontology, it allows greater flexibility for the repositories, as they do not need a single *provider* that needs be able to resolve all terms. Similarly, different repositories can use the same ontology, but use different *providers*. Note that this would not require changes to the data, as the data would only contains CURIEs, not the (provider-specific) IRIs.

The AIRR schema will provide a list of AIRR approved CURIE *prefixes* along with a list of at least one IRI *prefix* (i.e., replacement string) for each them. This list serves two purposes:

1. It provides a controlled namespace for CURIE *prefixes* used in the AIRR schema. For now, custom additions to or replacements of these *prefixes* in the schema is prohibited. This does not affect the ability of repositories to use such custom prefixes internally.
2. It simplifies resolution of CURIEs by non-repositories. The lists of IRI *prefixes* for each CURIE *prefix* should not be considered to be exhaustive. However, when using custom IRI *prefixes*, it must be ensured that they refer to the same ontology as the provider *prefixes*.

It should be explicitly noted that the IRI *prefix* list should not be interpreted as any kind of recommendation for certain *providers*. It is left up to users to decide how to resolve the resulting IRIs, e.g., via DNS/HTTP (if possible) or by using a *provider* of their choice.

Modifications to the AIRR schema

All changes to the AIRR schema that would be based on the sprint can currently be reviewed on Github in Pull Request #385. These changes are intended to be included into the next major release.

Clarifications

- Root nodes are specific to individual fields, not to an ontology. Therefore, NCBITAXON will use a root node of "Gnathostomata" for the annotations of the host species, but this would not be useful, e.g., if it would be used to annotate pathogenic organisms, which will require a top node at the apex of the hierarchy.
- The `labels` (previous: `values`) that are provided in the schema for ontology-based fields, should be considered an addition for convenience and not as being authoritative. Repositories or applications can choose to link synonyms to given concepts (e.g., "human" for "*Homo sapiens*") to simply search queries. Repositories further can provide such a synonym in the `label` field upon exporting data. However, repositories importing data should verify the correctness of `labels` that do not match the one provided by the ontology. Importing repositories must not be expected to allow for queries of `labels` other than those present in the ontology.

Annotation guidance

Note that this section is only a parking lot, the respective text will be moved into the AIRR Docs in the final version.

- Cells that come from Ficoll gradients should not be annotated as `PBMCs` as this is a sister node of `lymphocyte`. For the other sampling related fields, in nearly all cases venous blood (`UBERON:0013756`) will be the correct `tissue` and it should be used in the case of `sample_type`:`peripheral venous puncture`. However, if the mode of sampling is not specified, `blood` (`UBERON:0000178`) should be used instead. Also see https://github.com/airr-community/airr-standards/issues/242

Approved Ontologies

- Cell ontology (CL)
  - used in:

        - Cell subset (`cell_subset`, *Tissue and Cell Processing*)
    - default root node
        - label: `lymphocyte`
        - local id: `CL_0000542`
        - path: ‘‘
    - license: CC BY
    - latest release (as of 2020-05-20): 2020-03-02
    - repo: https://github.com/obophenotype/cell-ontology
    - maintainer: Alexander Diehl, Buffalo, NY, US (addiehl@buffalo.edu)
- Human disease ontology (DOID)
    - used in:
        - Diagnosis (`disease_diagnosis`, *Diagnosis*)
    - default root node
        - label: `disease`
        - local ID: `DOID:4`
        - path: `disease`
    - license: CC0
    - latest release (as of 2020-05-20): 2020-04-20
    - repo: https://github.com/DiseaseOntology/HumanDiseaseOntology
    - maintainer: Lynn Schriml, U Maryland, MD, US (lynn.schriml@gmail.com)
    - notes: Features ICD cross-reference
- NCBI organismal taxonomy (NCBITAXON)
    - used in:
        - Species (`species`, *Subject*)
        - Cell species (`cell_species`, *Tissue and Cell Processing*)
    - default root node
        - label: `Gnathostomata`
        - local ID: `7776`
        - path: `cellular organisms/Eukaryota/Opisthokonta/Metazoa/Eumetazoa/Bilateria/Deuterostomia/Chordata/Craniata/Vertebrata/Gnathostomata`
    - license: UMLS
    - latest release (as of 2020-05-20): 2020-04-18
    - repo: https://github.com/obophenotype/ncbitaxon
    - maintainer: NCBI (info@ncbi.nlm.nih.gov)
- NCI thesaurus (NCIT)
    - used in:

        - Study type (`study_type`, *Study*)
    - default root node
        - label: `Study`
        - local ID: `C63536`
        - path: `Activity/Clinical or Research Activity/ Research Activity/Study`
    - license: Public domain, credit of NCI is requested
    - repo: https://github.com/NCI-Thesaurus/thesaurus-obo-edition
    - latest release (as of 2020-05-20): 2020-05-04
    - maintainer: NCI (ncicbiitappssupport@mail.nih.gov)
- Units of measurement ontology (UO)
    - used in:
        - Age unit (`age_unit`, *Subject*)
    - default root node
        - label: `time unit`
        - local ID: `UO_0000003`
        - path: `unit/time unit`
    - license: CC BY (per Github repo)
    - repo: https://github.com/bio-ontology-research-group/unit-ontology
    - latest release (as of 2020-05-20): 2020-05-18
    - maintainer: unknown
- Uber-anatomy ontology (Uberon)
    - used in:
        - Tissue (`tissue`, *Sample*)
    - default root node
        - label: `multicellular anatomical structure`
        - local ID: `UBERON:0010000`
        - path: `/BFO_0000002/BFO_0000004/anatomical entity/material anatomical entity/anatomical structure/multicellular anatomical structure`
    - license: CC BY
    - repo: https://github.com/obophenotype/uberon
    - latest release (as of 2020-05-20): 2019-11-22
    - maintainer: Chris Mungall, LBL, CA, US (cjmungall@lbl.gov)

2.3.6 Schema Release Notes

Version 1.3.0: May 28, 2020

Version 1.3 schema release.

New Schema:

1. Introduced the `Repertoire` Schema for describing study meta data.
2. Introduced the PCRTarget Schema for describing primer target locations.
3. Introduced the SampleProcessing Schema for describing experimental processing steps for a sample.
4. Replaced the SoftwareProcessing schema with the DataProcessing schema.
5. Introduced experimental schema for clonal clusters, lineage trees, tree nodes, and cells as Clone, Tree, Node, and Cell objects, respectively.

General Updates:

1. Added multiple additional attributes to a large number of schema propertes as AIRR extension attributes in the `x-airr` field. The new `Attributes` object contains definitions for these `x-airr` field attributes.
2. Added the top level `required` property to all relevant schema objects.
3. Added the `title` attribute containing the short, descriptive name to all relevant schema object fields.
4. Added an `example` attribute containing an example data value to multiple schema object fields.

AIRR Data Commons API:

1. Added OpenAPI V2 specification (`specs/adc-api.yaml`) for AIRR Data Commons API major version 1.

Ontology Support:

1. Added `Ontology` and `CURIEResolution` objects to support ontologies.
2. Added vocabularies/ontologies as JSON string for: Cell subset, Target substrate, Library generation method, Complete sequences, Physical linkage of different loci.

Rearrangement Schema:

1. Added the `complete_vdj` field to annotate whether a V(D)J alignment was full length.
2. Added the `junction_length_aa` field defining the length of the junction amino acid sequence.
3. Added the `repertoire_id`, `sample_processing_id`, and `data_processing_id` fields to serve as linkers to the appropriate metadata objects.
4. Added a controlled vocabulary to the `locus` field: `IGH`, `IGI`, `IGK`, `IGL`, `TRA`, `TRB`, `TRD`, `TRG`.
5. Deprecated the `rearrangement_set_id` and `germline_database` fields.
6. Deprecated `rearrangement_id` field and made the `sequence_id` field be the primary unique identifer for a rearrangement record, both in files and data repositories.
7. Added support secondary D gene rearrangement through the additional fields: `d2_call`, `d2_score`, `d2_identity`, `d2_support`, `d2_cigar` `np3`, `np3_aa`, `np3_length`, `n3_length`, `p5d2_length`, `p3d2_length`, `d2_sequence_start`, `d2_sequence_end`, `d2_germline_start`, `d2_germline_start`, `d2_alignment_start`, `d2_alignment_end`, `d2_sequence_alignment`, `d2_sequence_alignment_aa`, `d2_germline_alignment`, `d2_germline_alignment_aa`.
8. Updated field definitions with more concise V(D)J call descriptions.

Alignment Schema:

1. Deprecated the `rearrangement_set_id` and `germline_database` fields.
2. Added the `data_processing_id` field.

Study Schema:

1. Added the `study_type` field containing an ontology defined term for the study design.

Subject Schema:

1. Deprecated the `organism` field in favor of the new `species` field.
2. Deprecated the `age` field.
3. Introduced age ranges: `age_min`, `age_max`, and `age_unit`.

Diagnosis Schema:

1. Changed the type of the `disease_diagnosis` field from `string` to `Ontology`.

Sample Schema:

1. Changed the type of the `tissue` field from `string` to `Ontology`.

CellProcessing Schema:

1. Changed the type of the `cell_subset` field from `string` to `Ontology`.
2. Introduced the `cell_species` field which denotes the species from which the analyzed cells originate.

NucleicAcidProcessing Schema:

1. Defined the `template_class` field as type `string`.
2. Added a controlled vocabulary the `library_generation_method` field.
3. Changed the controlled vocabulary terms of `complete_sequences`. Replacing `complete & untemplated` with `complete+untemplated` and adding `mixed`.
4. Added the `pcr_target` field referencing the new `PCRTarget` schema object.

SequencingRun Schema:

1. Added the `sequencing_run_id` field which serves as the object identifer field.
2. Added the `sequencing_files` field which links to the RawSequenceData schema objects defining the raw read data.

RawSequenceData Schema:

1. Added the `file_type` field defining the sequence file type. This field is a controlled vocabulary restricted to: `fasta`, `fastq`.
2. Added the `paired_read_length` field defining mate-pair read lengths.
3. Defined the `read_direction` and `paired_read_direction` fields as type `string`.

DataProcessing Schema:

1. Replaces the SoftwareProcessing object.
2. Added `data_processing_id`, `primary_annotation`, `data_processing_files`, `germline_database` and `analysis_provenance_id` fields.

Version 1.2.1: Oct 5, 2018

Minor patch release.

1. Schema gene vs segment terminology corrections
2. Added `Info` object
3. Updated `cell_subset` URL in AIRR schema

Version 1.2.0: Aug 18, 2018

Peer reviewed released of the Rearrangement schema.

1. Definition change for the coordinate fields of the Rearrangement and Alignment schema. Coordinates are now defined as 1-based closed intervals, instead of 0-based half-open intervals (as previously defined in v1.1 of the schema).
2. Removed foreign `study_id` fields
3. Introduced `keywords_study` field

Version 1.1.0: May 3, 2018

Initial public released of the Rearrangement and Alignment schemas.

1. Added `required` and `nullable` constrains to AIRR schema.
2. Schema definitions for MiAIRR attributes and ontology.
3. Introduction of an `x-airr` object indicating if field is required by MiAIRR.
4. Rename `rearrangement_set_id` to `data_processing_id`.
5. Rename `study_description` to `study_type`.
6. Added `physical_quantity` format.
7. Raw sequencing files into separate schema object.
8. Rename Attributes object.
9. Added `primary_annotation` and `repertoire_id`.
10. Added `diagnosis` to repertoire object.
11. Added ontology for `organism`.
12. Added more detailed specification of `sequencing_run`, `repertoire` and `rearrangement`.
13. Added repertoire schema.
14. Rename `definitions.yaml` to `airr-schema.yaml`.
15. Removed `c_call`, `c_score` and `c_cigar` from required as this is not typical reference aligner output.
16. Renamed `vdj_score`, `vdj_identity`, `vdj_evalue`, and `vdj_cigar` to `score`, `identity`, `evalue`, and `cigar`.
17. Added missing `c_identity` and `c_evalue` fields to `Rearrangement` spec.
18. Swapped order of *N* and *S* operators in CIGAR string.
19. Some description clean up for consistency in `Rearrangement` spec.

20. Remove repeated objects in `definitions.yaml`.
21. Added `Alignment` object to `definitions.yaml`.
22. Updated MiARR format consistency check TSV with junction change.
23. Changed definition from functional to productive.

Version 1.0.1: Jan 9, 2018

MiAIRR v1 official release and initial draft of Rearrangement and Alignment schemas.

2.4 Data Submission and Query

2.4.1 Data Submission Guides for AIRR-seq studies

There are multiple data repositories that accept submission of AIRR-seq datasets. Each provides different capabilities but all comply with the MiAIRR standard.

CAIRR Pipeline

Introduction: The CAIRR pipeline for submitting standards-compliant B and T cell receptor repertoire sequencing studies to the NCBI

AIRR sequencing (AIRR-seq) has tremendous potential to understand the dynamics of the immune repertoire in vaccinology, infectious disease, autoimmunity, and cancer biology. The adaptation of high-throughput sequencing (HTS) for AIRR (Adaptive Immune Receptor Repertoire) studies has made possible to characterize the AIRR at unprecedented depth and the outcome of such sequencing produces big data. Effective sharing of AIRR-seq big data could potentially reveal amazing scientific insights. The AIRR Community has proposed MiAIRR (Minimum information about an Adaptive Immune Receptor Repertoire Sequencing Experiment), a standard for reporting AIRR-seq studies. The MiAIRR standard has been implemented using the National Center for Biotechnology Information (NCBI) repositories. Submissions of AIRR-seq data to the NCBI repositories typically use a combination of web-based and flat-file templates and include only a minimal amount of terminology validation. As a result, AIRR-seq studies at the NCBI are often described using inconsistent terminologies, limiting scientists' ability to access, find, interoperate, and reuse the data sets and to understand how the experiments were performed. CEDAR (Center for Expanded Data Annotation and Retrieval) develops technologies involving the use of data standards and ontologies to improve metadata quality. In order to improve metadata quality and ease AIRR-seq study submission process, we have developed an AIRR-seq data submission pipeline named CEDAR-AIRR (CAIRR). CAIRR leverages CEDAR's technologies to: i) create web-based templates whose entries are controlled by ontology terms, ii) generate and validate metadata and iii) submit the ontology-linked metadata and sequence files (FASTQ) to the NCBI BioProject, BioSample, and Sequence Read Archive (SRA) databases. Thus, CAIRR provides a web-based metadata submission interface that supports compliance with MiAIRR standards. The interface enables ontology-based validation for several data elements, including: organism, disease, cell type and subtype, and tissue. This pipeline will facilitate the NCBI submission process and improve the metadata quality of AIRR-seq studies.

Submission Steps

The submission steps are described in the MiAIRR-to-NCBI Submission Manual: *Option 1. Submission via the CEDAR system (CAIRR submission pipeline)*. You will need a CEDAR system account; you can self-register at https://cedar.metadatacenter.org. You will also need the identifier of a BioProject already entered in the NCBI BioProject database.

Citing the MiAIRR Pipeline

Bukhari, Syed Ahmad Chan, Martin J. O'Connor, Marcos Martínez-Romero, Attila L. Egyedi, Debra Debra Willrett, John Graybeal, Mark A. Musen, Florian Rubelt, Kei H. Cheung, and Steven H. Kleinstein. The CAIRR pipeline for submitting standards-compliant B and T cell receptor repertoire sequencing studies to the NCBI. Frontiers in Immunology 9 (2018): 1877. DOI: 10.3389/fimmu.2018.01877

Tell Us About It

Please let us know how it went! If you are willing, we would love to have your comments in a short survey, it should just take 5 minutes or so.

We also welcome entry of issues and requests in our github repository issues, and emails can be sent to cedar-users@lists.stanford.edu. Both of these resources are publicly visible.

Support or Contact

Having trouble with NCBI submission process through our pipeline? Please email to Syed Ahmad Chan Bukhari or to Marcos Martínez-Romero and we will help you sort it out.

Introduction to VDJServer

VDJServer is a cloud-based analysis portal for immune repertoire sequence data that provides access to a suite of tools for a complete analysis workflow, including modules for preprocessing and quality control of sequence reads, V(D)J gene assignment, repertoire characterization, and repertoire comparison. VDJServer also provides sophisticated visualizations for exploratory analysis. It is accessible through a standard web browser via a graphical user interface designed for use by immunologists, clinicians, and bioinformatics researchers. VDJServer provides a data commons for public sharing of repertoire sequencing data, as well as private sharing of data between users.

- VDJServer website
- VDJServer Community Data Portal
- Email VDJServer for information about submission of your study data.

References

Introduction to iReceptor

iReceptor is a platform for storing, sharing, and exploring AIRR-seq data according to the AIRR Community standards.

- iReceptor Website (General information)
- iReceptor Gateway (AIRR Data Commons data query and federation)
- iReceptor Repositories (AIRR Data Commons repositories)
- iReceptor Turnkey GitHUb (Software)
- Email iReceptor (Contact).

References

2.4.2 Data Submission for Inferred Genes and Alleles

In 2017, The AIRR Community established the Inferred Allele Review Committee (IARC) to evaluate inferred alleles for inclusion in relevant germline databases. IARC has worked, together with colleagues at IMGT and the US National Institutes of Health, to establish a systematic submission and review process. OGRDB was created and designed to support that process, and provide a real-time record of affirmed sequences.

OGRDB - reference database of inferred immune receptor genes

In recent years it has become possible to sequence immune receptor repertoires (immunoglobulins and T cell receptors) at great depth. The accurate analysis of these repertoires requires a comprehensive understanding of the germline genes that give rise to the repertoire through V(D)J gene recombination.

Even for well-studied species such as humans and mice, our knowledge of allelic variation is incomplete. Identifying new immunoglobulin and T cell receptor polymorphisms from the genome using traditional methods is technically challenging, because of the complex sequence architecture and repetitive nature of these loci. More recently, methods have been developed to infer novel sequences and alleles from sequenced repertoires.

The Adaptive Immune Receptor Repertoire (AIRR) Community was formed to promote and share good practice in adaptive immune repertoire sequencing. In 2017, it established the Inferred Allele Review Committee (IARC) to evaluate inferred alleles for inclusion in relevant germline databases. IARC's work is outlined in more detail in a poster, which was presented at a Systems Immunology Workshop at the University of Surrey, England, in March 2018, and in a recent paper. IARC has worked, together with colleagues at IMGT and the US National Institute of Health, to establish a systematic submission and review process. OGRDB was created and designed to support that process, and provide a real-time record of affirmed sequences. Affirmed sequences will be listed under the Sequences tab above, and the submissions that underpin them will be found under the Submissions tab. You can make your own submissions by following the steps below.

How to submit your sequences

As a first step, IARC is now ready to review submissions of inferred human IGHV genes and alleles. These sequences may be novel, or may extend incomplete sequences currently in the IMGT reference directories. Researchers interested in submitting sequences should:

Submission of IARC gene inference data to NCBI

General outline

IARC submission currently follows a "INSDC first" approach, means that all sequence data related to the reported inference is REQUIRED to be properly deposited in a general purpose sequence repository before it is reviewed by IARC. The submitter needs to complete the initial steps of submission to one of the INSDC repositories. Upon submission to IARC, some of this data will be pulled in from NCBI (TODO: What kind of data can we actually pull down from INSDC?)

The aim of this procedure is to ensure that inferences reviewed by IARC are public and will remain available in the long run. It is however explicitly *not* the aim to provide data that deterministically will yield the same inference results.

Deposition of inferred gene data at NCBI

At the end of the deposition process there should be three types of records present at NCBI:

1. A single record containing the final and full-length inferred sequence. The record is deposited in one of the following:

   - Genbank: All inferences that have been performed on the submitters own data CAN be submitted as [???] to Genbank. Note that Genbank typically only holds data that has a physical correlate which is not necessarily true for inferred sequences. Nevertheless NCBI currently accepts this as a kind of consensus building if it is performed on your own data. The Genbank record MUST link to the `select set` record (see 3.) via the `DBLINK/DR` field. Genbank records will be publicly available independent of other publications. Note that the for Genbank, the `DBLINK` field does not appear to be available through the BankIt submission interface. You can use `Tbl2asn` and `Sequin`, and edit the `DBLINK` field manually (as "Sequence Read Archive" is not one of the options on the template creation page. A sample Genbank deposit can be found under accession MK321694.
   - TPA (Third-party annotation): A segment of Genbank dedicated inferences. Also the TPA record MUST link to the `select set` record (see 3.) via the `DBLINK/DR` field. Note that in contrast to Genbank, TPA does REQUIRE a peer-reviewed publication describing the details of the inference process before the record will be made publicly available. A sample TPA deposit can be found under accession BK01573.

   The format for both record types the Genbank format (link) with a standardized feature table (FT). Note that your initial submission MUST NOT contain any potential name for the gene as this will be assigned by IARC later on.

   TODO: Is there any metadata that should be provided into the GB record?

2. One or multiple SRA records containing all raw reads of the the respective sequencing run. Note that if you are performing inference using third-party data, these records MUST be submitted by the original owner of the data. These record type will typically be present before the other. The metadata annotation of the records SHOULD be MiAIRR compliant [Rubelt et al.].

3. One or multiple SRA records containing the `select set` of reads from (2). The aim of these records is to document the number, quality, coverage and diversity of the reads in a dataset that _potentially_ support the inference. This means that the `select set` SHOULD be a superset of the reads that support the inference. It is NOT REQUIRED that inference tools deterministically return the inferred allele upon being fed with the `select set`. Generation of the `select set` from the complete set is described below. When submitting the `select set` to SRA the metadata context, i.e. the original links to project, sample and (if possible) experiment) SHOULD be maintained. Reads originating from multiple subjects or samples MUST NOT be pooled into a single new entry. The new record SHOULD be titled "Reads from <original_run_accession> supporting inference of Homo sapiens immunoglobulin heavy chain variable gene" and contain a design description, e.g., "Experimental workflow as described in original SRA/ENA record [<run_accession>]. Gene inference was performed using <software+version+parameters>. The reported reads were selected based on <selection_criteria>."

NOTE: It is reasonably likely, in the short term, that you will encounter questions from the SRA/ENA/Genbank staff about the nature of these deposits. If so, you can respond that they are made as part of a community effort to document novel alleles with an emphasis on transparency in data provenance. You can link to the IARC page and note that we worked together with IMGT and Genbank/TPA staff in designing this procedure.

Generating the `select set`

Below is the current procedure describing how to generate a `select set` using general purpose tools. This procedure was designed in a rather generic fashion so that it is easy to implement and does NOT REQUIRE inference tools to provide their own mechanisms. Note that it is currently assumed that the procedure is not fully deterministic, i.e. the `select set` cannot simply be generated using the complete read data and the inferred sequence, as there are additional filter criteria that apply. In addition the `select set` SHOULD not be subject to any modifications that are not listed below. This includes UMI-based consensus building or other aggregation steps that are not fully transparent to a third-party.

1. Assemble paired-end reads. The two reads MUST overlap. Recommended tool: PandaSeq
2. Perform PHRED filtering that is equivalent to the one performed by inference pipeline. Recommended tool: Immcantation suite
3. Perform a *blastn* search using the data from (2.) as query and bp 1-312 of the inferred gene as reference library. Require matches to be full-length and >99.6% ID. Record all matching read ID. Recommended tool: NCBI BLAST
4. Select the reads with the read ID found in (3.) from the original unmerged FASTQs. Note that each `select set` MUST be derived from a single donor and sample. Recommended tool: Christian's cryptic extractor script
5. Submit the `select set` to SRA

Submit the inferred sequences to IARC via OGRDB, following the

- OGRDB Submission Guide

Additional information is available at the

- OGRDB Website

References

2.4.3 Data Query and Download from the AIRR Data Commons

Submission of AIRR-seq datasets to public data repositories means that other researchers can query, download and reuse that data for novel analyses.

AIRR Data Commons

The AIRR Data Commons is a network of distributed repositories that store AIRR-seq data and adhere to the AIRR Community standards. We define the AIRR Data Commons as consisting of the set of repositories that both:

- Adhere to the AIRR Common Repositories Working Group recommendations for promoting, sharing, and use of AIRR-seq data.
- Implement the *ADC API* as a programmatic mechanism to access that data.

More information on repositories in the AIRR Data Commons and how to query these repositories can be found on the AIRR Data Commons page:

AIRR Data Commons

The use of high-throughput sequencing for profiling B-cell and T-cell receptors has resulted in a rapid increase in data generation. It is timely, therefore, for the Adaptive Immune Receptor Repertoire (AIRR) community to establish a clear set of community-accepted data and metadata standards; analytical tools; and policies and practices for infrastructure to support data deposit, curation, storage, and use. Such actions are in accordance with international funder and journal policies that promote data deposition and data sharing – at a minimum, data on which scientific publications are based should be made available immediately on publication. Data deposit in publicly accessible databases ensures that published results may be validated. Such deposition also facilitates reuse of data for the generation of new hypotheses and new knowledge.

The AIRR Common Repository Working Group (CRWG) has developed a set of recommendations that promote the deposit, sharing, and use of AIRR sequence data. These recommendations were refined following community discussions at the AIRR 2016 and 2017 Community Meetings and were approved through a vote by the AIRR Community at the AIRR Community Meeting in December 2017. Updates to these recommendations have continued, with the latest set of Recommendations ratified at the AIRR Community meeting in May 2019.

In May 2020, the AIRR Community released the first verion of the AIRR Data Commons Application Programming Interface (ADC API), a specification for programmatic access to query and download AIRR-seq data from repositories that adhere to the AIRR Standards. We define the AIRR Data Commons as consisting of the set of repositories that:

- adhere to the CRWG recommendations for promoting, sharing, and use of AIRR-seq data, and
- that implement the ADC API as a programmatic mechanism to access that data.

This page provides a central location for the community to discover resources that belong to the AIRR Data Commons.

AIRR Data Commons Repositories

These data repositories all implement the AIRR Data Commons (ADC) API programmatic access to query and download AIRR-seq data.

- *iReceptor Public Archive*
- *VDJServer Community Data Portal*

Querying the AIRR Data Commons

Each of the repositories above can be queried directly using the *ADC API*. In addition, the following tools and platforms implement web based user interfaces that use the ADC API to query repositories in the AIRR Data Commons:

- *iReceptor Gateway*

There are *query and analysis use cases* and *a set of example queries* available for the AIRR Data Commons and the ADC API.

Other Public AIRR-Seq Repositories

There are additional data repositories that provide access to AIRR-seq data but which did not implement the ADC API for programmatic access. Information about some of these repositories are provided in a B-T.CR forum post.

Germline Gene Inference and Usage

- *OGRDB* provides a list of alleles affirmed by the AIRR Community's Inferred Allele Review Committee, together with supporting information.
- VDJbase provides gene usage information derived from a growing base of AIRR-seq repertoires, including inferred genotypes and haplotypes.

2.5 Software

2.5.1 AIRR Python Reference Library

The `airr` reference library provides basic functions and classes for interacting with AIRR Community Data Representation Standards, including tools for read, write and validation.

API Reference

Rearrangement Interface

`airr.`**`read_rearrangement`** (*filename*, *validate=False*, *debug=False*)

Open an iterator to read an AIRR rearrangements file

Parameters

- **`file`** (*`str`*) – path to the input file.
- **`validate`** (*`bool`*) – whether to validate data as it is read, raising a ValidationError exception in the event of an error.
- **`debug`** (*`bool`*) – debug flag. If True print debugging information to standard error.

Returns iterable reader class.

Return type *airr.io.RearrangementReader*

`airr.`**`create_rearrangement`** (*filename*, *fields=None*, *debug=False*)

Create an empty AIRR rearrangements file writer

Parameters

- **`filename`** (*`str`*) – output file path.
- **`fields`** (*`list`*) – additional non-required fields to add to the output.
- **`debug`** (*`bool`*) – debug flag. If True print debugging information to standard error.

Returns open writer class.

Return type *airr.io.RearrangementWriter*

`airr.`**`derive_rearrangement`** (*out_filename*, *in_filename*, *fields=None*, *debug=False*)

Create an empty AIRR rearrangements file with fields derived from an existing file

Parameters

- **`out_filename`** (*`str`*) – output file path.
- **`in_filename`** (*`str`*) – existing file to derive fields from.
- **`fields`** (*`list`*) – additional non-required fields to add to the output.
- **`debug`** (*`bool`*) – debug flag. If True print debugging information to standard error.

Returns open writer class.

Return type *airr.io.RearrangementWriter*

`airr.`**`load_rearrangement`** (*filename*, *validate=False*, *debug=False*)

Load the contents of an AIRR rearrangements file into a data frame

Parameters

- **filename** (*str*) – input file path.
- **validate** (*bool*) – whether to validate data as it is read, raising a ValidationError exception in the event of an error.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns Rearrangement records as rows of a data frame.

Return type pandas.DataFrame

airr.**dump_rearrangement** (*dataframe*, *filename*, *debug=False*)
Write the contents of a data frame to an AIRR rearrangements file

Parameters

- **dataframe** (*pandas.DataFrame*) – data frame of rearrangement data.
- **filename** (*str*) – output file path.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns True if the file is written without error.

Return type bool

airr.**merge_rearrangement** (*out_filename*, *in_filenames*, *drop=False*, *debug=False*)
Merge one or more AIRR rearrangements files

Parameters

- **out_filename** (*str*) – output file path.
- **in_filenames** (*list*) – list of input files to merge.
- **drop** (*bool*) – drop flag. If True then drop fields that do not exist in all input files, otherwise combine fields from all input files.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns True if files were successfully merged, otherwise False.

Return type bool

airr.**validate_rearrangement** (*filename*, *debug=False*)
Validates an AIRR rearrangements file

Parameters

- **filename** (*str*) – path of the file to validate.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns True if files passed validation, otherwise False.

Return type bool

Repertoire Interface

airr.**load_repertoire** (*filename*, *validate=False*, *debug=False*)
Load an AIRR repertoire metadata file

Parameters

- **filename** (*str*) – path to the input file.

- **validate** (*bool*) – whether to validate data as it is read, raising a ValidationError exception in the event of an error.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns list of Repertoire dictionaries.

Return type list

airr.**write_repertoire**(*filename*, *repertoires*, *info=None*, *debug=False*)

Write an AIRR repertoire metadata file

Parameters

- **file** (*str*) – path to the output file.
- **repertoires** (*list*) – array of repertoire objects.
- **info** (*object*) – info object to write. Will write current AIRR Schema info if not specified.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns True if the file is written without error.

Return type bool

airr.**validate_repertoire**(*filename*, *debug=False*)

Validates an AIRR repertoire metadata file

Parameters

- **filename** (*str*) – path of the file to validate.
- **debug** (*bool*) – debug flag. If True print debugging information to standard error.

Returns True if files passed validation, otherwise False.

Return type bool

airr.**repertoire_template**()

Return a blank repertoire object from the template. This object has the complete structure with all of the fields and all values set to None or empty string.

Returns empty repertoire object.

Return type object

Classes

class airr.io.**RearrangementReader**(*handle*, *base=1*, *validate=False*, *debug=False*)

Iterator for reading Rearrangement objects in TSV format

fields

field names in the input Rearrangement file.

Type list

external_fields

list of fields in the input file that are not part of the Rearrangement definition.

Type list

__init__(*handle*, *base=1*, *validate=False*, *debug=False*)

Initialization

Parameters

- **handle** (*file*) – file handle of the open Rearrangement file.
- **base** (*int*) – one of 0 or 1 specifying the coordinate schema in the input file. If 1, then the file is assumed to contain 1-based closed intervals that will be converted to python style 0-based half-open intervals for known fields. If 0, then values will be unchanged.
- **validate** (*bool*) – perform validation. If True then basic validation will be performed will reading the data. A ValidationError exception will be raised if an error is found.
- **debug** (*bool*) – debug state. If True prints debug information.

Returns reader object.

Return type *airr.io.RearrangementReader*

`__iter__`()

Iterator initializer

Returns airr.io.RearrangementReader

`__next__`()

Next method

Returns parsed Rearrangement data.

Return type dict

close()

Closes the Rearrangement file

next()

Next method

class airr.io.**RearrangementWriter**(*handle*, *fields=None*, *base=1*, *debug=False*)

Writer class for Rearrangement objects in TSV format

fields

field names in the output Rearrangement file.

Type list

external_fields

list of fields in the output file that are not part of the Rearrangement definition.

Type list

`__init__`(*handle*, *fields=None*, *base=1*, *debug=False*)

Initialization

Parameters

- **handle** (*file*) – file handle of the open Rearrangements file.
- **fields** (*list*) – list of non-required fields to add. May include fields undefined by the schema.
- **base** (*int*) – one of 0 or 1 specifying the coordinate schema in the output file. Data provided to the write is assumed to be in python style 0-based half-open intervals. If 1, then data will be converted to 1-based closed intervals for known fields before writing. If 0, then values will be unchanged.
- **debug** (*bool*) – debug state. If True prints debug information.

Returns writer object.

Return type *airr.io.RearrangementWriter*

close()

Closes the Rearrangement file

write(*row*)

Write a row to the Rearrangement file

Parameters **row** (*dict*) – row to write.

class airr.schema.**Schema**(*definition*)

AIRR schema definitions

properties

field definitions.

Type collections.OrderedDict

info

schema info.

Type collections.OrderedDict

required

list of mandatory fields.

Type list

optional

list of non-required fields.

Type list

false_values

accepted string values for False.

Type list

true_values

accepted values for True.

Type list

from_bool(*value*, *validate=False*)

Converts a boolean to a string

Parameters

- **value** (*bool*) – logical value.
- **validate** (*bool*) – when True raise a ValidationError for an invalid value. Otherwise, set invalid values to None.

Returns conversion of True or False or 'T' or 'F'.

Return type str

Raises airr.ValidationError – raised if value is invalid when validate is set True.

spec(*field*)

Get the properties for a field

Parameters **name** (*str*) – field name.

Returns definition for the field.

Return type collections.OrderedDict

to_bool(*value*, *validate=False*)
Convert a string to a boolean

Parameters

- **value** (*str*) – logical value as a string.
- **validate** (*bool*) – when True raise a ValidationError for an invalid value. Otherwise, set invalid values to None.

Returns conversion of the string to True or False.

Return type bool

Raises `airr.ValidationError` – raised if value is invalid when validate is set True.

to_float(*value*, *validate=False*)
Converts a string to a float

Parameters

- **value** (*str*) – float value as a string.
- **validate** (*bool*) – when True raise a ValidationError for an invalid value. Otherwise, set invalid values to None.

Returns conversion of the string to a float.

Return type float

Raises `airr.ValidationError` – raised if value is invalid when validate is set True.

to_int(*value*, *validate=False*)
Converts a string to an integer

Parameters

- **value** (*str*) – integer value as a string.
- **validate** (*bool*) – when True raise a ValidationError for an invalid value. Otherwise, set invalid values to None.

Returns conversion of the string to an integer.

Return type int

Raises `airr.ValidationError` – raised if value is invalid when validate is set True.

type(*field*)
Get the type for a field

Parameters **name** (*str*) – field name.

Returns the type definition for the field

Return type str

validate_header(*header*)
Validate header against the schema

Parameters **header** (*list*) – list of header fields.

Returns True if a ValidationError exception is not raised.

Return type bool

Raises `airr.ValidationError` – raised if header fails validation.

validate_object(*obj*, *missing=True*, *nonairr=True*, *context=None*)
Validate Repertoire object data against schema

Parameters

- **obj** (*dict*) – dictionary containing a single repertoire object.
- **missing** (*bool*) – provides warnings for missing optional fields.
- **(bool** (*nonairr*) – provides warning for non-AIRR fields that cannot be validated.
- **context** (*string*) – used by recursion to indicate place in object hierarchy

Returns True if a ValidationError exception is not raised.

Return type bool

Raises `airr.ValidationError` – raised if object fails validation.

validate_row(*row*)
Validate Rearrangements row data against schema

Parameters **row** (*dict*) – dictionary containing a single record.

Returns True if a ValidationError exception is not raised.

Return type bool

Raises `airr.ValidationError` – raised if row fails validation.

Schema

airr.schema.**AlignmentSchema Schema object for the Alignment definition**
AIRR schema definitions

airr.schema.**properties**
field definitions.

Type collections.OrderedDict

airr.schema.**info**
schema info.

Type collections.OrderedDict

airr.schema.**required**
list of mandatory fields.

Type list

airr.schema.**optional**
list of non-required fields.

Type list

airr.schema.**false_values**
accepted string values for False.

Type list

airr.schema.**true_values**
accepted values for True.

Type list

airr.schema.**RearrangementSchema Schema object for the Rearrangement definition**

AIRR schema definitions

airr.schema.**properties**

field definitions.

Type collections.OrderedDict

airr.schema.**info**

schema info.

Type collections.OrderedDict

airr.schema.**required**

list of mandatory fields.

Type list

airr.schema.**optional**

list of non-required fields.

Type list

airr.schema.**false_values**

accepted string values for False.

Type list

airr.schema.**true_values**

accepted values for True.

Type list

airr.schema.**RepertoireSchema Schema object for the Repertoire definition**

AIRR schema definitions

airr.schema.**properties**

field definitions.

Type collections.OrderedDict

airr.schema.**info**

schema info.

Type collections.OrderedDict

airr.schema.**required**

list of mandatory fields.

Type list

airr.schema.**optional**

list of non-required fields.

Type list

airr.schema.**false_values**

accepted string values for False.

Type list

airr.schema.**true_values**

accepted values for True.

Type list

Commandline Tools

airr-tools

AIRR Community Standards utility commands.

```
usage: airr-tools [-h] [--version]  ...
```

-h, --help
show this help message and exit

--version
show program’s version number and exit

airr-tools merge

Merge AIRR rearrangement files.

```
usage: airr-tools merge [--version] [-h] -o OUT_FILE [--drop] -a AIRR_FILES
                        [AIRR_FILES ...]
```

--version
show program’s version number and exit

-h, --help
show this help message and exit

-o <out_file>
Output file name.

--drop
If specified, drop fields that do not exist in all input files. Otherwise, include all columns in all files and fill missing data with empty strings.

-a <airr_files>
A list of AIRR rearrangement files.

airr-tools validate

Validate AIRR files.

```
usage: airr-tools validate [--version] [-h]  ...
```

--version
show program’s version number and exit

-h, --help
show this help message and exit

airr-tools validate rearrangement

Validate AIRR rearrangement files.

```
usage: airr-tools validate rearrangement [--version] [-h] -a AIRR_FILES
                                         [AIRR_FILES ...]
```

--version
show program's version number and exit

-h, --help
show this help message and exit

-a <airr_files>
A list of AIRR rearrangement files.

airr-tools validate repertoire

Validate AIRR repertoire metadata files.

```
usage: airr-tools validate repertoire [--version] [-h] -a AIRR_FILES
                                      [AIRR_FILES ...]
```

--version
show program's version number and exit

-h, --help
show this help message and exit

-a <airr_files>
A list of AIRR repertoire metadata files.

Python Library Release Notes

Version 1.3.0: May 30, 2020

1. Updated schema set to v1.3.
2. Added `load_repertoire`, `write_repertoire`, and `validate_repertoire` to `airr.interface` to read, write and validate Repertoire metadata, respectively.
3. Added `repertoire_template` to `airr.interface` which will return a complete repertoire object where all fields have `null` values.
4. Added `validate_object` to `airr.schema` that will validate a single repertoire object against the schema.
5. Extended the `airr-tools` commandline program to validate both rearrangement and repertoire files.

Version 1.2.1: October 5, 2018

1. Fixed a bug in the python reference library causing start coordinate values to be empty in some cases when writing data.

Version 1.2.0: August 17, 2018

1. Updated schema set to v1.2.
2. Several improvements to the `validate_rearrangement` function.
3. Changed behavior of all *airr.interface* functions to accept a file path (string) to a single Rearrangement TSV, instead of requiring a file handle as input.

4. Added `base` argument to `RearrangementReader` and `RearrangementWriter` to support optional conversion of 1-based closed intervals in the TSV to python-style 0-based half-open intervals. Defaults to conversion.
5. Added the custom exception `ValidationError` for handling validation checks.
6. Added the `validate` argument to `RearrangementReader` which will raise a `ValidationError` exception when reading files with missing required fields or invalid values for known field types.
7. Added `validate` argument to all type conversion methods in `Schema`, which will now raise a `ValidationError` exception for value that cannot be converted when set to `True`. When set `False` (default), the previous behavior of assigning `None` as the converted value is retained.
8. Added `validate_header` and `validate_row` methods to `Schema` and removed validations methods from `RearrangementReader`.
9. Removed automatic closure of file handle upon reaching the iterator end in `RearrangementReader`.

Version 1.1.0: May 1, 2018

Initial release.

Installation

Install in the usual manner from PyPI:

```
> pip3 install airr --user
```

Or from the downloaded source code directory:

```
> python3 setup.py install --user
```

Quick Start

Reading AIRR Repertoire metadata files

The `airr` package contains functions to read and write AIRR repertoire metadata files. The file format is either YAML or JSON, and the package provides a light wrapper over the standard parsers. The file needs a `json`, `yaml`, or `yml` file extension so that the proper parser is utilized. All of the repertoires are loaded into memory at once and no streaming interface is provided:

```
import airr

# Load the repertoires
data = airr.load_repertoire('input.airr.json')
for rep in data['Repertoire']:
    print(rep)
```

Why are the repertoires in a list versus in a dictionary keyed by the `repertoire_id`? There are two primary reasons for this. First, the `repertoire_id` might not have been assigned yet. Some systems might allow MiAIRR metadata to be entered but the `repertoire_id` is assigned to that data later by another process. Without the `repertoire_id`, the data could not be stored in a dictionary. Secondly, the list allows the repertoire data to have a default ordering. If you know that the repertoires all have a unique `repertoire_id` then you can quickly create a dictionary object using a comprehension:

```
rep_dict = { obj['repertoire_id'] : obj for obj in data['Repertoire'] }
```

Writing AIRR Repertoire metadata files

Writing AIRR repertoire metadata is also a light wrapper over standard YAML or JSON parsers. The `airr` library provides a function to create a blank repertoire object in the appropriate format with all of the required fields. As with the load function, the complete list of repertoires are written at once, there is no streaming interface:

```
import airr

# Create some blank repertoire objects in a list
reps = []
for i in range(5):
    reps.append(airr.repertoire_template())

# Write the repertoires
airr.write_repertoire('output.airr.json', reps)
```

Reading AIRR Rearrangement TSV files

The `airr` package contains functions to read and write AIRR rearrangement files as either iterables or pandas data frames. The usage is straightforward, as the file format is a typical tab delimited file, but the package performs some additional validation and type conversion beyond using a standard CSV reader:

```
import airr

# Create an iteratable that returns a dictionary for each row
reader = airr.read_rearrangement('input.tsv')
for row in reader: print(row)

# Load the entire file into a pandas data frame
df = airr.load_rearrangement('input.tsv')
```

Writing AIRR formatted files

Similar to the read operations, write functions are provided for either creating a writer class to perform row-wise output or writing the entire contents of a pandas data frame to a file. Again, usage is straightforward with the `airr` output functions simply performing some type conversion and field ordering operations:

```
import airr

# Create a writer class for iterative row output
writer = airr.create_rearrangement('output.tsv')
for row in reader:  writer.write(row)

# Write an entire pandas data frame to a file
airr.dump_rearrangement(df, 'file.tsv')
```

Validating AIRR data files

The airr package can validate repertoire and rearrangement data files to insure that they contain all required fields and that the fields types match the AIRR Schema. This can be done using the airr-tools command line program or the validate functions in the library can be called:

```
# Validate a rearrangement file
airr-tools validate rearrangement -a input.tsv

# Validate a repertoire metadata file
airr-tools validate repertoire -a input.airr.json
```

Combining Repertoire metadata and Rearrangement files

The airr package does not keep track of which repertoire metadata files are associated with rearrangement files, so users will need to handle those associations themselves. However, in the data, the repertoire_id field forms the link. The typical usage is that a program is going to perform some computation on the rearrangements, and it needs access to the repertoire metadata as part of the computation logic. This example code shows the basic framework for doing that, in this case doing gender specific computation:

```
import airr

# Load the repertoires
data = airr.load_repertoire('input.airr.json')

# Put repertoires in dictionary keyed by repertoire_id
rep_dict = { obj['repertoire_id'] : obj for obj in data['Repertoire'] }

# Create an iteratable for rearrangement data
reader = airr.read_rearrangement('input.tsv')
for row in reader:
    # get repertoire metadata with this rearrangement
    rep = rep_dict[row['repertoire_id']]

    # check the gender
    if rep['subject']['sex'] == 'male':
        # do male specific computation
    elif rep['subject']['sex'] == 'female':
        # do female specific computation
    else:
        # do other specific computation
```

2.5.2 AIRR Data Representation Reference Library

airr is an R package for working with data formatted according to the AIRR Data Representation schemas. It includes the full set of schema definitions along with simple functions for read, write and validation.

Usage Vignette

Introduction

Since the use of High-throughput sequencing (HTS) was first introduced to analyze immunoglobulin (B-cell receptor, antibody) and T-cell receptor repertoires (Freeman et al, 2009; Robins et al, 2009; Weinstein et al, 2009), the increasing number of studies making use of this technique has produced enormous amounts of data and there exists a pressing need to develop and adopt common standards, protocols, and policies for generating and sharing data sets. The Adaptive Immune Receptor Repertoire (AIRR) Community formed in 2015 to address this challenge (Breden et al, 2017) and has stablished the set of minimal metadata elements (MiAIRR) required for describing published AIRR datasets (Rubelt et al, 2017) as well as file formats to represent this data in a machine-readable form. The airr R package provide read, write and validation of data following the AIRR Data Representation schemas. This vignette provides a set of simple use examples.

AIRR Data Representation Standards

The AIRR Community's recommendations for a minimal set of metadata that should be used to describe an AIRR-seq data set when published or deposited in a AIRR-compliant public repository are described in Rubelt et al, 2017. The primary aim of this effort is to make published AIRR datasets FAIR (findable, accessible, interoperable, reusable); with sufficient detail such that a person skilled in the art of AIRR sequencing and data analysis will be able to reproduce the experiment and data analyses that were performed.

Following this principles, V(D)J reference alignment annotations are saved in standard tab-delimited files (TSV) with associated metadata provided in accompanying YAML formatted files. The column names and field names in these files have been defined by the AIRR Data Representation Working Group using a controlled vocabulary of standardized terms and types to refer to each piece of information.

Reading AIRR formatted files

The airr package contains the function read_rearrangement to read and validate files containing AIRR Rearrangement records, where a Rearrangement record describes the collection of optimal annotations on a single sequence that has undergone V(D)J reference alignment. The usage is straightforward, as the file format is a typical tabulated file. The argument that needs attention is base, with possible values "0" and "1". base denotes the starting index for positional fields in the input file. Positional fields are those that contain alignment coordinates and names ending in "_start" and "_end". If the input file is using 1-based closed intervals (R style), as defined by the standard, then positional fields will not be modified under the default setting of base="1". If the input file is using 0-based coordinates with half-open intervals (python style), then positional fields may be converted to 1-based closed intervals using the argument base="0".

```
library(airr)

example_data <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")
basename(example_data)
```

```
## [1] "rearrangement-example.tsv.gz"
```

```
airr_rearrangement <- read_rearrangement(example_data)
class(airr_rearrangement)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
head(airr_rearrangement)
```

```
## # A tibble: 6 x 33
##   sequence_id sequence rev_comp productive vj_in_frame stop_codon v_call d_call j_
↪call c_call sequence_alignm... germline_alignm... junction junction_aa v_cigar d_
↪cigar
##   <chr>       <chr>    <lgl>    <lgl>      <lgl>       <lgl>      <chr>  <chr>
↪<chr>  <chr>  <chr>              <chr>              <chr>    <chr>       <chr>   <chr>
```

(continues on next page)

(continued from previous page)

```
## 1 SRR765688.... NNNNNNN... FALSE    TRUE       TRUE        FALSE      IGHV2...␣
↪IGHD5... IGHJ4... IGHG   .................. CAGATCACCTTGAAG... TGTGCAC...␣
↪CAHSAGWLPD... 20S56N... 274S5N...
## 2 SRR765688.... NNNNNNN... FALSE    TRUE       TRUE        FALSE      IGHV5...␣
↪IGHD3... IGHJ6... IGHG   .................. GAGGTGCAGCTGGTG... TGTGCGA...␣
↪CARHGLYGCD... 20S40N... 305S29...
## 3 SRR765688.... NNNNNNN... FALSE    TRUE       TRUE        FALSE      IGHV7...␣
↪IGHD3... IGHJ4... IGHG   .................. CAGGTGCAGCTGGTG... TGTGCGA...␣
↪CAREERRSSG... 20S33N... 293S13...
## 4 SRR765688.... NNNNNNN... FALSE    TRUE       TRUE        FALSE      IGHV7...␣
↪IGHD3... IGHJ6... IGHG   .................. CAGGTGCAGCTGGTG... TGTGCGA...␣
↪CAREGYYFDT... 20S33N... 290S9N...
## 5 SRR765688.... NNNNNNN... FALSE    TRUE       TRUE        FALSE      IGHV7...␣
↪IGHD1... IGHJ6... IGHG   .................. CAGGTGCAGCTGGTG... TGTGCGA...␣
↪CARDSGGMDVW 20S33N... 283S4N...
## 6 SRR765688.... NNNNNNN... FALSE    FALSE      TRUE        TRUE       IGHV2...␣
↪IGHD2... IGHJ4... IGHA   .................. CAGATCACCTTGAAG... TGTGTCC...␣
↪CVLSRRLGDS... 20S56N... 273S12...
## # ... with 17 more variables: j_cigar <chr>, v_sequence_start <int>, v_sequence_
↪end <int>, v_germline_start <int>, v_germline_end <int>, d_sequence_start <int>,
## #   d_sequence_end <int>, d_germline_start <int>, d_germline_end <int>, j_sequence_
↪start <int>, j_sequence_end <int>, j_germline_start <int>, j_germline_end <int>,
## #   junction_length <int>, np1_length <int>, np2_length <int>, duplicate_count
↪<int>
```

Writing AIRR formatted files

The airr package contains the function write_rearrangement to write Rearrangement records to the AIRR TSV format.

```
out_file <- file.path(tempdir(), "airr_out.tsv")
write_rearrangement(airr_rearrangement, out_file)
```

References

1. Breden, F., E. T. Luning Prak, B. Peters, F. Rubelt, C. A. Schramm, C. E. Busse, J. A. Vander Heiden, et al. 2017. Reproducibility and Reuse of Adaptive Immune Receptor Repertoire Data. *Front Immunol* 8: 1418.

2. Freeman, J. D., R. L. Warren, J. R. Webb, B. H. Nelson, and R. A. Holt. 2009. Profiling the T-cell receptor beta-chain repertoire by massively parallel sequencing. *Genome Res* 19 (10): 1817-24.

3. Robins, H. S., P. V. Campregher, S. K. Srivastava, A. Wacher, C. J. Turtle, O. Kahsai, S. R. Riddell, E. H. Warren, and C. S. Carlson. 2009. Comprehensive assessment of T-cell receptor beta-chain diversity in alphabeta T cells. *Blood* 114 (19): 4099-4107.

4. Rubelt, F., C. E. Busse, S. A. C. Bukhari, J. P. Burckert, E. Mariotti-Ferrandiz, L. G. Cowell, C. T. Watson, et al. 2017. Adaptive Immune Receptor Repertoire Community recommendations for sharing immune-repertoire sequencing data. *Nat Immunol* 18 (12): 1274-8.

5. Weinstein, J. A., N. Jiang, R. A. White, D. S. Fisher, and S. R. Quake. 2009. High-throughput sequencing of the zebrafish antibody repertoire. *Science* 324 (5928): 807-10.

Reference Topics

read_airr

Read an AIRR TSV

Description

`read_airr` reads a TSV containing AIRR records.

Usage

```
read_airr(file, base = c("1", "0"), schema = RearrangementSchema, ...)
```

```
read_rearrangement(file, base = c("1", "0"), ...)
```

```
read_alignment(file, base = c("1", "0"), ...)
```

Arguments

file input file path.

base starting index for positional fields in the input file. If `"1"`, then these fields will not be modified. If `"0"`, then fields ending in `"_start"` and `"_end"` are 0-based half-open intervals (python style) in the input file and will be converted to 1-based closed-intervals (R style).

schema `Schema` object defining the output format.

... additional arguments to pass to read_delim.

Value

A data.frame of the TSV file with appropriate type and position conversion for fields defined in the specification.

Details

`read_rearrangement` reads an AIRR TSV containing Rearrangement data.

`read_alignment` reads an AIRR TSV containing Alignment data.

Examples

```
# Get path to the rearrangement-example file
file <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")

# Load data file
df <- read_rearrangement(file)
```

See also

See Schema for the AIRR schema object definition. See write_airr for writing AIRR data.

write_airr

Write an AIRR TSV

Description

write_airr writes a TSV containing AIRR formatted records.

Usage

```
write_airr(data, file, base = c("1", "0"), schema = RearrangementSchema, ...)
```

```
write_rearrangement(data, file, base = c("1", "0"), ...)
```

```
write_alignment(data, file, base = c("1", "0"), ...)
```

Arguments

data data.frame of Rearrangement data.

file output file name.

base starting index for positional fields in the output file. Fields in the input data are assumed to be 1-based closed-intervals (R style). If "1", then these fields will not be modified. If "0", then fields ending in _start and _end will be converted to 0-based half-open intervals (python style) in the output file.

schema Schema object defining the output format.

... additional arguments to pass to write_delim.

Details

write_rearrangement writes a data.frame containing AIRR Rearrangement data to TSV.

write_alignment writes a data.frame containing AIRR Alignment data to TSV.

Examples

```
# Get path to the rearrangement-example file
file <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")

# Load data file
df <- read_rearrangement(file)
```

(continues on next page)

(continued from previous page)

```
# Write a Rearrangement data file
outfile <- file.path(tempdir(), "output.tsv")
write_rearrangement(df, outfile)
```

See also

See Schema for the AIRR schema object definition. See read_airr for reading to AIRR files.

validate_airr

Validate AIRR data

Description

`validate_airr` validates compliance of the contents of a data.frame to the AIRR data standards.

Usage

```
validate_airr(data, schema = RearrangementSchema)
```

Arguments

data data.frame to validate.

schema `Schema` object defining the data standard.

Value

Returns `TRUE` if the input `data` is compliant and `FALSE` if not.

Examples

```
# Get path to the rearrangement-example file
file <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")

# Load data file
df <- read_rearrangement(file)

# Validate a data.frame against the Rearrangement schema
validate_airr(df, schema=RearrangementSchema)
```

```
[1] TRUE
```

load_schema

Load a schema definition

Description

load_schema loads an AIRR object definition from the internal definition set.

Usage

```
load_schema(definition)
```

Arguments

definition name of the schema definition.

Value

A Schema object for the definition.

Details

Valid definitions include:

- "Rearrangement"
- "Alignment"
- "Study"
- "Subject"
- "Diagnosis"
- "Sample"
- "CellProcessing"
- "NucleicAcidProcessing"
- "RawSequenceData"
- "SoftwareProcessing"

Examples

```
# Load the Rearrangement definition
schema <- load_schema("Rearrangement")

# Load the Alignment definition
schema <- load_schema("Alignment")
```

See also

See Schema for the return object.

Schema-class

S4 class defining an AIRR standard schema

Description

`Schema` defines a common data structure for AIRR Data Representation standards.

Usage

```
"names"(x)
```

```
"["(x, i)
```

```
"$"(x, name)
```

```
AlignmentSchema
```

```
RearrangementSchema
```

Arguments

x `Schema` object.

i field name.

name field name.

Format

A `Schema` object.

An object of class `Schema` of length 1.

An object of class `Schema` of length 1.

Details

The following predefined Schema objects are defined:

`AlignmentSchema`: AIRR Alignment `Schema`.

`RearrangementSchema`: AIRR Rearrangement `Schema`.

Slots

required character vector of required fields.

optional character vector of non-required fields.

properties list of field definitions.

info list schema information.

See also

See load_schema for loading a Schema from the definition set. See read_airr, write_airr and validate_airr schema operators.

ExampleData

Example AIRR data

Description

Example data files compliant with the the AIRR Data Representation standards.

Format

extdata/rearrangement-example.tsv.gz: Rearrangement TSV file.

Examples

```
# Get path to the rearrangement-example file
file <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")

# Load data file
df <- read_rearrangement(file)
```

R Library Release Notes

Version 1.3.0: May 26, 2020

1. Updated schema set to v1.3.
2. Added info slot to Schema object containing general schema information.

Version 1.2.0: August 17, 2018

1. Updated schema set to v1.2.
2. Changed defaults to base="1" for read and write functions.

3. Updated example TSV file with coordinate changes, addition of `germline_alignment` data and simplification of `sequence_id` values.

Version 1.1.0: May 1, 2018

Initial release.

Download & Installation

To install the latest release from CRAN:

```
install.packages("airr")
```

To build from the source code, first install the build dependencies:

```
install.packages(c("devtools", "roxygen2"))
```

To install the latest development code via devtools:

```
library(devtools)
install_github("airr-community/airr-standards/lang/R@master")
```

Note, using `install_github` will not build the documentation. To generate the documentation, clone the repository, and then build as normal using the following R commands from the package root `lang/R`:

```
library(devtools)
install_deps(dependencies=T)
document()
install()
```

Dependencies

Imports: methods, readr, stats, stringi, yaml
Suggests: knitr, rmarkdown, testthat

Authors

Jason Vander Heiden (aut, cre)
Susanna Marquez (aut)
Scott Christley (aut)
AIRR Community (cph)

License

CC BY 4.0

2.5.3 ADC API Reference Implementation

The AIRR Community provides a reference implementation for an ADC API service. The reference implementation can be utilized for any number of tasks. For example, a data repository might use the source code as a starting point for their own implementation and can compare the behaviour of their service against the reference. Another example is a tool developer, who wishes to use the API, can setup a local data repository so they can develop and test their tool before sending API requests across the internet to remote data repositories. While the reference implementation is functionally complete, it has minimal security and no optimizations for large data so it should not be used directly for production systems.

The reference implementation consists of three GitHub repositories: adc-api, adc-api-js-mongodb, and adc-api-mongodb-repository. The three repositories correspond to the top-level service composition (adc-api), a JavaScript web service that responds to API requests and queries a MongoDB database (adc-api-js-mongodb), and a MongoDB database for holding AIRR-seq data (adc-api-mongodb-repository). Docker and docker-compose are used to provide a consistent deployment environment and compose the multiple components together into a single service. Complete documentation for configuring and deploying the reference implementation is available in the adc-api repository.

2.6 Community Resources

2.6.1 Resources and Tools Supporting AIRR Standards

Applications Supporting the Rearrangement Schema

The following list of software tools and databases support the TSV format of v1.2 of the *AIRR Rearrangement schema*.

Software	Version	Support	Reference
AIRR Python Library	1.2	Input, output and validation	Vander Heiden et al. Front Immunol, 2018.
AIRR R Library	1.2	Input, output and validation	Vander Heiden et al. Front Immunol, 2018.
Decombinator	4.0.1	Output	Oakes et al. Front Immunol, 2017.
IMGT/V-QUEST	3.5.16	Output	Giudicelli et al. Cold Spring Harb Protoc, 2011.
IgBLAST	1.11	Output	Ye et al. Nucleic Acids Res, 2013.
IGoR	TBD	Input and output	Marcou et al. Nat Commun, 2018.
Immcantation:Change-O	0.4.2	Input, output and conversion	Gupta & Vander Heiden et al. Bioinformatics, 2015.
ImmuneDB	0.24.0	Output	Rosenfeld et al. Front Immunol, 2018.
iReceptor	2	Input and output	Corrie et al. Immunol Rev, 2018.
MiXCR	2.2.1	Output	Bolotin et al. Nat Methods, 2015.
OLGA	TBD	Input and output	Sethna et al. Bioinformatics, 2019.
Partis	TBD	Output	Ralph & Matsen. PLoS Comput Biol, 2016.
SONAR	3	Output	Schramm et al. Front Immunol, 2016.
TRIgS	2	Input	Lees & Shepherd. J Immunol Res, 2015.
VDJServer	1.2.0	Input and output	Christley et al. Front Immunol, 2018
Vidjil-algo	2018.1	Output	Giraud et al. BMC Genomics, 2014.
Vidjil Web Platform	TBD	Input and conversion	Duez et al. PLoS ONE, 2016.

AIRR Data Commons Repositories

These data repositories all implement the AIRR Data Commons (ADC) API programmatic access to query and download AIRR-seq data.

- *iReceptor Public Archive*
- *VDJServer Community Data Portal*

2.6.2 Useful Websites for the AIRR Community

- The Antibody Society
- The AIRR Community of the Antibody Society
- B-T.CR Forum
- The AIRR Community GitHub

- The AIRR Standards GitHub Repository
- The AIRR Community Docker Hub

2.7 Appendix A: Key Terms

The following table provides definitions for terms and acronyms relevant to this documentation.

Term	Definition
ADC	AIRR Data Commons
AIRR	Adaptive Immune Receptor Repertoire
AIRR-C	AIRR Community
API	Application Programming Interface
CAIRR	CEDAR AIRR
CEDAR	Center for Expanded Data Annotation and Retrieval
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MiAIRR	Minimal Information about an Adaptive Immune Receptor Repertoire study
REST	Representational State Transfer
TSV	Tab Separated Values
URL	Universal Resource Locator
YAML	YAML Ain't Markup Language

2.8 References

Bibliography

[LIGMDB_V12] IMGT-ONTOLOGY definitions. <http://www.imgt.org/ligmdb/label#JUNCTION>

[INSDC_FT] The DDBJ/ENA/GenBank Feature Table Definition. <http://www.insdc.org/documents/feature-table>

[ENA_MANUAL] European Nucleotide Archive Annotated/Assembled Sequences User Manual. <http://ftp.ebi.ac.uk/pub/databases/ena/sequence/release/doc/usrman.txt>

[GENBANK_FF] GenBank Flat File Format. <https://ftp.ncbi.nih.gov/genbank/gbrel.txt>

[GENBANK_SR] GenBank Sample Record. <https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html>

[INSDC_XREF] Controlled vocabulary for `/db_xref` qualifier. <http://www.insdc.org/documents/dbxref-qualifier-vocabulary>

[NCBI_NBK47528] SRA Handbook. <https://www.ncbi.nlm.nih.gov/books/NBK47528/>

[RFC3987] Internationalized Resource Identifiers (IRIs). DOI:10.17487/RFC3987

[Christley_2018] Christley S *et al*. VDJServer: A Cloud-Based Analysis Portal and Data Commons for Immune Repertoire Sequences and Rearrangements. Front Immunol 9:976 (2018) DOI: 10.3389/fimmu.2018.00976

[Corrie_2018] Corrie *et al*. iReceptor: A platform for querying and analyzing antibody/B-cell and T-cell receptor repertoire data across federated repositories. Immunol Rev. 2018 Jul;284(1):24-41. DOI: 10.1111/imr.12666

[Ohlin_2019] Ohlin M *et al*. Inferred Allelic Variants of Immunoglobulin Receptor Genes: A System for Their Evaluation, Documentation, and Naming. Front Immunol 10:435 (2019) DOI: 10.3389/fimmu.2019.00435

[Breden_2017] Breden F *et al*. Reproducibility and Reuse of Adaptive Immune Receptor Repertoire Data. Front Immunol 8:1418 (2017). DOI: 10.3389/fimmu.2017.01418

[Christley_2020] Christley S *et al*. The ADC API: a web API for the programmatic query of the AIRR Data Commons. Front in Big Data (2020). DOI: 10.3389/fdata.2020.00022

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels DOI: 10.17487/RFC2119

[Rubelt_2017] Rubelt F *et al*. AIRR Community Recommendations for Sharing Immune Repertoire Sequencing Data. Nat Immunol 18:1274 (2017). DOI: 10.1038/ni.3873

[VanderHeiden_2018] Vander Heiden JA *et al*. AIRR Community Standardized Representations for Annotated Immune Repertoires. Front Immunol 9:2206 (2018). DOI: 10.3389/fimmu.2018.02206

[Wilkinson_2016] Wilkinson MD *et al*. The FAIR Guiding Principles for scientific data management and stewardship. Sci Data 3:160018 (2016). DOI: 10.1038/sdata.2016.18

[Zenodo_1185414] Release archive of the AIRR Standards repository. (2015-2020). DOI: 10.5281/zenodo.1185414

Index