airr-standards Documentation

Release 2.0

AIRR Community

Jun 08, 2026

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

ONE

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 study reporting that describes minimal information about AIRR datasets, including sample collection and data processing information.
- Data representation and 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

TWO

TABLE OF CONTENTS

2.1 Getting Started

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

2.1.1 AIRR data standards

- Gather experimental and analysis information about your study to conform to the *MiAIRR* standard (minimal information about adaptive immune receptor repertoires).
- Schema, definitions and file formats for the *AIRR Data Standards*. The AIRR Data Model defines the structure and relationship for AIRR data elements.

2.1.2 AIRR data query and submission

- *Query* publicly available AIRR-seq studies in the *AIRR Data Commons*.
- *Submission* of your study data to a public repository.

2.1.3 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.
- *AIRR Data Commons API* provides programmatic access to query and download AIRR-seq data.
- *Resources and tools* that support the AIRR Standards.

2.2 Introduction to MiAIRR

An early objective of the Adaptive Immune Receptor Repertoire (AIRR) Community was the development of metadata standards for AIRR sequencing dataset submissions. This effort was led by the AIRR Community Minimal Standards Working Group, which disbanded upon completion of its task. To enhance reproducibility, standardize quality control, and facilitate data deposition in a shared repository, the AIRR Community has adopted six high-level data sets as published in 2017 [Rubelt_2017]. These data sets 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 in the section *MiAIRR Data Elements* (`Download as TSV`).

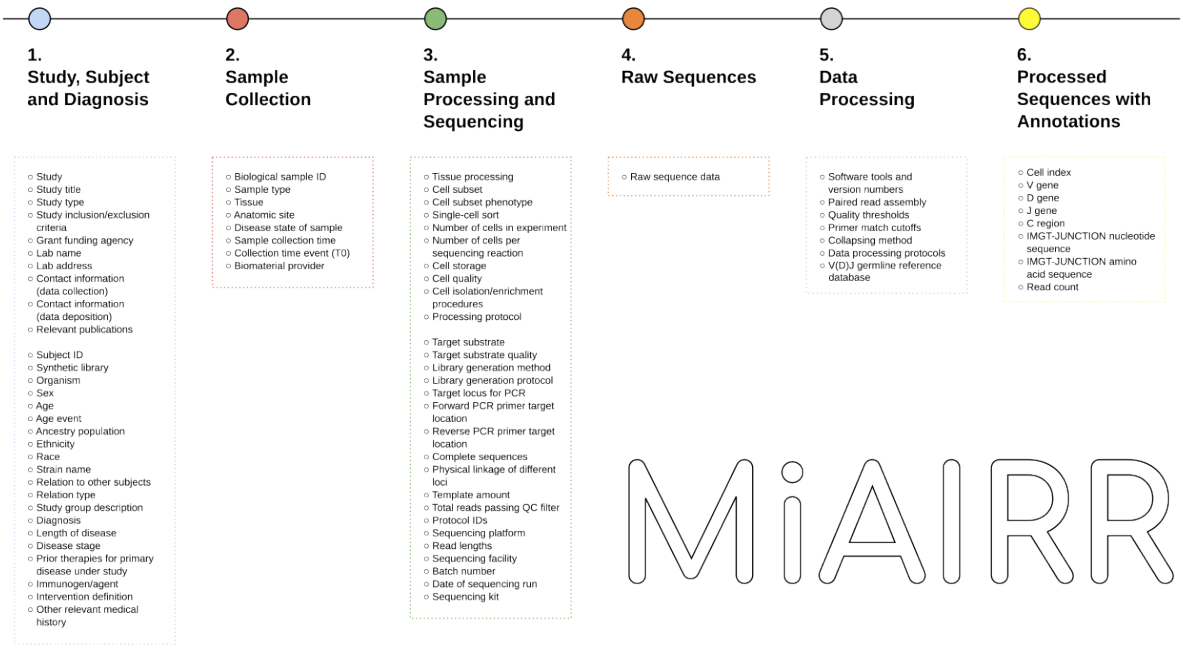

Fig. 1: Schema of MiAIRR data sets and the individual data elements of each set.

2.2.1 Topics

MiAIRR Data Elements

The AIRR Community has adopted six high-level sets that will guide the publication, curation and sharing of AIRR-Seq data and metadata:

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

`Download as TSV`.

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / stud	Study ID `study_id`	string *free text*	important	Unique ID assigned by study registry such as one of the International Nucleotide Sequence Database Collaboration (INSDC) repositories.	PRJNA001

continues 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, label: Study}}*	important	Type of study design	id: NCIT:C15197, label: Case-Control Study
1 / study	Study inclusion/exclusion criteria `inclusion_ex`	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	Contributors `contributors`	array of *Contributor* **	essential	List of individuals who contributed to the study. Note that these are not necessarily identical with the authors on an associated manuscript or other scholarly communication. Further note that typically at least the three CRediT contributor roles "supervision", "investigation" and "data curation" should be assigned. The coresponding author should be listed last.	
1 / study	Relevant publications `pub_ids`	array of string **	important	Array of publications describing the rationale and/or outcome of the study as an array of CURIE objects such as a DOI or Pubmed ID. Where more than one publication is given, if there is a primary publication for the study it should come first.	['PMID:29144493', 'DOI:10.1038/ni.3873']

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / study	Keywords for study `keywords_stu`	array of string *Controlled vocabulary: contains_ig, contains_tr, contains_paired_ch contains_schema_r contains_schema_c contains_schema_c contains_schema_r contains_schema_c contains_schema_r*	important	Keywords describing properties of one or more data sets in a study. "contains_schema" keywords indicate that the study contains data objects from the AIRR Schema of that type (Rearrangement, Clone, Cell, Receptor) while the other keywords indicate that the study design considers the type of data indicated (e.g. it is possible to have a study that "contains_paired_chain" but does not "contains_schema_cell").	['contains_ig', 'contains_schema_rearrangement', 'contains_schema_clone', 'contains_schema_cell']
1 / subject	Subject ID `subject_id`	string *free text*	important	Subject ID assigned by submitter, unique within study. If possible, a persistent subject ID linked to an INSDC or similar repository study should be used.	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:7 label: Gnathostomata}}*	essential	Binomial designation of subject's species	id: NCBITAXON:9606, label: Homo sapiens
1 / subject	Sex `sex`	string *Controlled vocabulary: male, female, pooled, hermaphrodite, intersex*	important	Biological sex of subject	female
1 / subject	`age`	*TimeInterval* **	important	Age of subject expressed as a time interval. If singular time point then min == max in the time interval.	

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
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
1 / subject	Ancestry population `ancestry_popu`	*Ontology* *Ontology: { top_node: { id: GAZ:00000448, label: geographic location}}*	important	Broad geographic origin of ancestry (continent)	id: GAZ:00000459, label: South America
1 / subject	`location_birt`	*Ontology* *Ontology: { top_node: { id: GAZ:00000448, label: geographic location}}*	important	Self-reported location of birth of the subject, preferred granularity is country-level	id: GAZ:00002939, label: Poland
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_subje`	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

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / diagnosis and intervention	Study group description `study_group_`	string *free text*	important	Designation of study arm to which the subject is assigned to	control
1 / diagnosis and intervention	Diagnosis timepoint `diagnosis_ti`	*TimePoint* **	important	Time point for the diagnosis	OrderedDict([('label', 'Study enrollment'), ('value', 60), ('unit', OrderedDict([('id', 'UO:0000033'), ('label', 'day')]))])
1 / diagnosis and intervention	Diagnosis `disease_diag`	*Ontology* *Ontology: { top_node: { id: DOID:4, label: disease}}*	important	Diagnosis of subject	id: DOID:9538, label: multiple myeloma
1 / diagnosis and intervention	Length of disease `disease_leng`	*TimeQuantity* **	important	Time duration between initial diagnosis and current intervention	OrderedDict([('quantity', 23), ('unit', OrderedDict([('id', 'UO:0000035'), ('label', 'month')]))])

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
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
1 / diagnosis and intervention	Intervention definition `intervention`	string *free text*	important	Description of intervention	systemic chemotherapy, 6 cycles, 1.25 mg/m2

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
1 / diagnosis and intervention	Other relevant medical history `medical_hist`	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. If possible, a persistent sample ID linked to INSDC or similar repository study should be used.	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:0010 label: multicellular anatomical structure}}*	important	The actual tissue sampled, e.g. lymph node, liver, peripheral blood	id: UBERON:0002371, label: bone marrow
2 / sample	Anatomic site `anatomic_sit`	string *free text*	important	The anatomic location of the tissue, e.g. Inguinal, femur	Iliac crest
2 / sample	Disease state of sample `disease_stat`	string *free text*	important	Histopathologic evaluation of the sample	Tumor infiltration
2 / sample	Sample collection time `collection_t`	*TimePoint* **	important	Time point at which sample was taken, relative to *label* event	OrderedDict([('label', 'Primary vaccination'), ('value', 14), ('unit', OrderedDict([('id', 'UO:0000033'), ('label', 'day')]))])

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
2 / sample	Sample collection location `collection_l`	*Ontology Ontology: { top_node: { id: GAZ:00000448, label: geographic location}}*	important	Location where the sample was taken, preferred granularity is country-level	id: GAZ:00002939, label: Poland
2 / sample	Biomaterial provider `biomaterial_`	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_proce`	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, label: lymphocyte}}*	important	Commonly-used designation of isolated cell population	id: CL:0000972, label: class switched memory B cell
3 / process (cell	Cell subset phenotype `cell_phenoty`	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 annotation `cell_label`	string *free text*	defined	Free text cell type annotation. Primarily used for annotating cell types that are not provided in the Cell Ontology.	age-associated B cell
3 / process (cell	Cell species `cell_species`	*Ontology Ontology: { top_node: { id: NCBITAXON:7 label: Gnathostomata}}*	defined	Binomial designation of the species from which the analyzed cells originate. Typically, this value should be identical to *species*, in which case it SHOULD NOT be set explicitly. However, 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, label: Homo sapiens
3 / process (cell	Single-cell sort `single_cell`	boolean *true* | *false*	important	TRUE if single cells were isolated into separate compartments	

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
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_re`	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 cryopreserved between isolation and further processing	True
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_isolati`	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_process`	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_cla`	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_qua`	string *free text*	important	Description and results of the quality control performed on the template material	RIN 9.2

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (nucleic acid	Template amount `template_amo`	PhysicalQuantity **	important	Amount of template that went into the process	OrderedDict([('quantity', 1000), ('unit', OrderedDict([('id', 'UO:0000024'), ('label', 'nanogram')]))])
3 / process (nucleic acid	Library generation method `library_gene`	string *Controlled vocabulary: PCR, RT(RHP)+PCR, RT(oligo-dT)+PCR, RT(oligo-dT)+TS+PCR, RT(oligo-dT)+TS(UMI)+ RT(specific)+P(RT(specific)+TS RT(specific)+TS RT(specific+UM RT(specific+UM RT(specific)+TS other*	essential	Generic type of library generation	RT(oligo-dT)+TS(UMI)+PCR
3 / process (nucleic acid	Library generation protocol `library_gene`	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_gene`	string *free text*	important	When using a library generation protocol from a commercial provider, provide the protocol version number	v2.1 (2016-09-15)

continues 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_seq`	string *Controlled vocabulary: partial, complete, complete+untempla 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

continues 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_lin`	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_l`	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_`	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_`	string *free text*	important	Position of the most proximal nucleotide templated by the reverse primer or primer mix	IGHG, +57

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
3 / process (sequering)	Batch number `sequencing_r`	string *free text*	important	ID of sequencing run assigned by the sequencing facility	160101_M01234
3 / process (sequering)	Total reads passing QC filter `total_reads_`	integer *positive integer*	important	Number of usable reads for analysis	10365118
3 / process (sequering)	Sequencing platform `sequencing_p`	string *free text*	important	Designation of sequencing instrument used	Alumina LoSeq 1000
3 / process (sequering)	Sequencing facility `sequencing_f`	string *free text*	important	Name and address of sequencing facility	Seqs-R-Us, Vancouver, BC, Canada
3 / process (sequering)	Date of sequencing run `sequencing_r`	string *free text*	important	Date of sequencing run	2016-12-16
3 / process (sequering)	Sequencing kit `sequencing_k`	string *free text*	important	Name, manufacturer, order and lot numbers of sequencing kit	FullSeq 600, Alumina, #M123456C0, 789G1HK
4 / data (raw read	Raw sequencing data persistent identifier `sequencing_d`	string *free text*	important	Persistent identifier of raw data stored in an archive (e.g. INSDC run ID). Data archive should be identified in the CURIE prefix.	SRA:SRR11610494
4 / data (raw read	Raw sequencing data file type `file_type`	string *Controlled vocabulary: fasta, fastq*	important	File format for the raw reads or sequences	

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
5 / process (computationa	Primer match cutoffs `primer_match`	string *free text*	important	How primers were identified in the sequences, were they removed/masked/etc?	Hamming distance <= 2
5 / process (computationa	Collapsing method `collapsing_m`	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 (computationa	Data processing protocols `data_process`	string *free text*	important	General description of how QC is performed	Data was processed using [...]
5 / process (computationa	V(D)J germline reference database `germline_dat`	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
6 / data (processe sequer	V gene with allele `v_call`	string *free text*	important	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).	IGHV4-59*01
6 / data (processe sequer	D gene with allele `d_call`	string *free text*	important	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).	IGHD3-10*01
6 / data (processe sequer	J gene with allele `j_call`	string *free text*	important	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).	IGHJ4*02

continues on next page

Table 1 – continued from previous page

Set / Subset	Designation / Field	Type / Format	Level	Definition	Example
6 / data (processe-quer	C region `c_call`	string *free text*	important	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).	IGHG1*01
6 / data (processe-quer	IMGT-JUNCTION nucleotide sequence `junction`	string *free text*	important	Junction region nucleotide sequence, where the junction is defined as the CDR3 plus the two flanking conserved codons.	TGTGCAAGAGCGGGAGTTTACGACGGATATACTATGGACTACTGG
6 / data (processe-quer	IMGT-JUNCTION amino acid sequence `junction_aa`	string *free text*	important	Amino acid translation of the junction.	CARAGVYDGYTMDYW
6 / data (processe-quer	Read count `duplicate_co`	integer *positive integer*	important	Copy number or number of duplicate observations for the query sequence. For example, the number of identical reads observed for this sequence.	123
6 / data (processe-quer	Cell index `cell_id`	string *free text*	important	Identifier defining the cell of origin for the query sequence.	W06_046_091

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_	library_generat	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)+PC	RT-PCR using oligo-dT primers
	RT(oligo-dT)+TS	5'-RACE PCR (i.e. RT is followed by a template switch (TS) step) using oligo-dT primers
	RT(oligo-dT)+TS	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)+PC	RT-PCR using transcript-specific primers
	RT(specific)+TS	5'-RACE PCR using transcript- specific primers
	RT(specific)+TS	5'-RACE PCR using transcript- specific primers and template switch primers containing UMIs
	RT(specific+UMI	RT-PCR using transcript-specific primers containing UMIs (i.e., the 3' end is UMI-coded)
	RT(specific+UMI	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

*_pcr_primer_target_location

The fields `forward_pcr_primer_target_location` and `reverse_pcr_primer_target_location` describe the location of the innermost primer used for a given locus. This information is critical to determine which positions of a sequence reflect a biological process and which are an artifact of the experiment.

The terms "proximal" and "distal" in the field description refer to the start of the respective V or C gene. Using human *TRAC* and a hypothetical primer as an example:

```
1   5    10   15   20   25   30   35   40   45   50   55  60   position (+1 = start of␣
↪exon 1)
|...|....|....|....|....|....|....|....|....|....|....|....|
                   TGCCGTGTACCAGCTGAGAG                        TRAC primer (reverse␣
↪complement)
ATATCCAGAACCCTGACCCTGCCGTGTACCAGCTGAGAGACTCTAAATCCAGTGACAAGT   Chr. 14:22,547,506-22,547,
↪565 (GRCh38)
                  ^^                  ^
                [a][b]               [c]                       markers
```

In this case:

- +19 (marker `[a]`) is the most distal untemplated nucleotide
- +20 (marker `[b]`) is the most proximal templated nucleotide
- +39 (marker `[c]`) is the most distal templated nucleotide

cell_subset

The `cell_subset` field is ontology-controlled, i.e., if present, it MUST refer to a Cell Ontology (CL) term via its `id` field. The field SHOULD NOT be used for values other than `lymphocyte` (CL:0000542) and its descendents. The reasoning behind this is that rearrangements of IG and TR loci are typically confined to this population, so that other nodes, do not provide further information. In addition, the field SHOULD only be used if the subset has been purified to a level that is comparable to flow cytometric cell sorting.

- In general, the provided annotation MUST NOT contradict the experimentally determined phenotype. E.g., if the experiment shows that the population is CD27+ a term that is explicitly defined in CL as CD27- MUST NOT be used.
- However, this does not mean that all markers listed in the description of an ontology term need to be confirmed, as long as the existing information is considered sufficient for classification and not contradictory (see above).
- In case the experimentally isolated cells to not match any "leaf" term, e.g., due to the isolation of multiple populations that contradict the definitions, the general advice is to move up the CL hierarchy to the most distal term that is no-contradictory. In this case, `cell_phenotype` should be used to define the markers that were used experimentally.
- Note that ontology-controlled fields allow exactly one term. Therefore, mixtures of defined cell populations either need to be demultiplexed, or – if this is not possible – use the last (i.e,, most distal) common term of all cell populations involved. Again, `cell_phenotype` can be used to provide the markers used in the experiment.

Specific Use Cases and Experimental Setups

Peripheral Blood Mononuclear Cells (PBMCs)

PBMCs are frequently used starting material for AIRR-seq studies in humans and are prepared by a density-gradient centrifugation using Ficoll. As they constitute a mixture of myeloid and lymphoid cells, the following points should be taken into consideration when annotating experiments using PBMCs:

- The `cell_population` and `cell_phenotype` fields should be `NULL` as PBMCs are neither sufficiently pure nor do they exclusively contain cells of the lymphocytic lineage (see *cell_subset*).
- Note that while Cell Ontology does provide a term `peripheral blood mononuclear cell` (CL:2000001), this is a sister node of `lymphocyte` (CL:0000542) and therefore outside of the current specification.
- The typical annotation for PBMC is therefore as follows:

```
sample_type:"peripheral venous puncture"
tissue:
   label:"venous blood"
   id:"UBERON:0013756"
tissue_processing:"Ficoll gradient"
cell_subset:NULL
cell_phenotype:NULL
```

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 that species. Importantly, it MUST NOT encode the phage vector, the bacterial host or a 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.

10X Chromium

The current 10X V(D)J Kits (07/2020, Rev. G) perform a fully nested PCR, in which only the reverse primers (i.e., complementary to the constant region) are Ig/TCR specific, while the forward primers anneal to the sequence of the template switch primer. For the purpose of annotation, this is considered a gene-specific amplification, therefore such experiments SHOULD be annotated as follows:

- `single_cell`: MUST be `true`
- `library_generation_method`: SHOULD be `RT(specific)+TS(UMI)+PCR`
- `pcr_target` MAY contain multiple entries, one for each locus that is potentially amplified. Within each entry (i.e., each `PCRTarget` object) the following annotations SHOULD be provided:
  - `pcr_target_locus`: The locus described by this object, using the controlled vocabulary defined in the AIRR schema. Note that each object can only describe one locus, multiple loci require multiple `PCRTarget` objects.
  - `forward_pcr_primer_target_location`: `NULL` (as it cannot be reliably determined.
  - `reverse_pcr_primer_target_location`: Locus and position according to the respective set of reverse primers.

Geolocation information

For questions regarding population genetics, the information about the membership of a given individual in a (potentially) distinct population group is of high interest. However, no reasonably complete ontology exists for the concept of "ethnicity", which in addition is limited to humans (not to mention the challenge of assigning consistent values to this property). Therefore, the AIRR Schema uses annotation of the location of birth of an individual and the location where a given sample was taken as imperfect but at least operationalizable substitute.

The respective properties `Subject.location_birth` and `Sample.collection_location` are ontology controlled and expect values that are geographic locations. The provided value SHOULD allow to resolve at least the country-level information of the annotated location. Higher granularity SHOULD only be provided, if this does not exposed the data subject to the risk of being re-identified via based on or assisted by this information.

2.3 AIRR Data Standards

AIRR Data Standards 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.

2.3.1 Schema Definitions

This document describes the AIRR Data Model. It begins with an overview of the structure and semantics of the `Repertoire` schema, including best practices for documenting data processing, principles for linking related data, and definitions of key concepts such as `Repertoire` and `Rearrangement`. This is followed by a specification of the file format and a detailed description of individual `Repertoire` fields.

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 and, ideally, to a specific sample, with 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 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.

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 appropriate 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 appropriate Repertoire object and then sample_processing_id should be used to determine the appropriate 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.

RepertoireFilter Schema

As a Repertoire corresponds to a discrete biological unit, it will often be the case that an experiment or analysis will span multiple Repertoires. Common examples include comparing individuals with and without a particular diagnosis or tracking repertoire evolution across a time series. Conversely, a researcher may sometimes be interested in only a specific subset of a Repertoire such as "productive rearrangements for IGHV4". All of these cases can be represented using an array of RepertoireFilters and contained in a RepertoireGroup.

A RepertoireFilter incorporates its underlying Repertoires by reference to their repertoire_ids and thus retains the ability to access all of the associated MiAIRR metadata. The RepertoireFilter also describes the selection criteria for the included repertoires and how they have been filtered by building a query equivalent to one that would be used in the ADC API.

RepertoireGroups can be associated with the same study as the underlying Repertoires or a new one, as appropriate.

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.

Schema Field Definitions

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_descri`	string	optional, nullable	Generic repertoire description
`repertoire_type`	string	optional, nullable	Repertoire type (source). Most often the type should be "observed," meaning it corresponds to an actual physical sample that was sequenced. Other allowed values are "simulated," i.e. the Rearrangements were generated in silico and there is no linked Subject/Sample metadata, and "inferred" for Rearrangements that are phylogenetically reconstructed from observed sequences. Inferred Repertoires should point to the same Subject/Sample metadata as the corresponding physical Repertoires.
`study`	*Study*	required	Study object
`subject`	*Subject*	required	Subject object
`sample`	array of SampleProcessing	required	List of Sample Processing objects
`data_processing`	array of *DataProcessing*	required	List of Data Processing objects

Repertoire Filter Fields

Download as TSV

Name	Type	Attributes	Definition
repertoire_id	string	optional	Identifier to the repertoire
repertoire_descri	string	optional, nullable	Description of this repertoire within the group
time_point	*TimePoint*	optional, nullable	Time point designation for this repertoire within the group
filter	object	optional, nullable	A JSON object describing how this Repertoire was filtered using the same structure as an ADC API query

Study Fields

Download as TSV

Name	Type	Attributes	Definition
study_id	string	required, identifier, nullable	Unique ID assigned by study registry such as one of the International Nucleotide Sequence Database Collaboration (INSDC) repositories.
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_exclusi	string	required, nullable	List of criteria for inclusion/exclusion for the study
grants	string	required, nullable	Funding agencies and grant numbers
contributors	array of *Contributor*	required	List of individuals who contributed to the study. Note that these are not necessarily identical with the authors on an associated manuscript or other scholarly communication. Further note that typically at least the three CRediT contributor roles "supervision", "investigation" and "data curation" should be assigned. The coresponding author should be listed last.
study_contact	string	DEPRECATED	Full contact information of the contact persons for this study This should include an e-mail address and a persistent identifier such as an ORCID ID.
collected_by	string	DEPRECATED	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 and a persistent identifier such as an ORCID ID.
lab_name	string	DEPRECATED	Department of data collector
lab_address	string	DEPRECATED	Institution and institutional address of data collector
submitted_by	string	DEPRECATED	Full contact information of the data depositor, i.e., the person submitting the data to a repository. This should include an e-mail address and a persistent identifier such as an ORCID ID. This is supposed to be a short-lived and technical role until the submission is relased.
pub_ids	array of string	required, nullable	Array of publications describing the rationale and/or outcome of the study as an array of CURIE objects such as a DOI or Pubmed ID. Where more than one publication is given, if there is a primary publication for the study it should come first.
keywords_study	array of string	required, nullable	Keywords describing properties of one or more data sets in a study. "contains_schema" keywords indicate that the study contains data objects from the AIRR Schema of that type (Rearrangement, Clone, Cell, Receptor) while the other keywords indicate that the study design considers the type of data indicated (e.g. it is possible to have a study that "contains_paired_chain" but does not "contains_schema_cell").
adc_publish_date	string	optional, nullable	Date the study was first published in the AIRR Data Commons.
adc_update_date	string	optional, nullable	Date the study data was updated in the AIRR Data Commons.

Subject Fields

Download as TSV

Name	Type	Attributes	Definition
subject_id	string	required, identifier, nullable	Subject ID assigned by submitter, unique within study. If possible, a persistent subject ID linked to an INSDC or similar repository study should be used.
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	*TimeInterval*	required, nullable	Age of subject expressed as a time interval. If singular time point then min == max in the time interval.
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_min	number	DEPRECATED	
age_max	number	DEPRECATED	
age_unit	*Ontology*	DEPRECATED	
ancestry_populati	*Ontology*	required, nullable	Broad geographic origin of ancestry (continent)
location_birth	*Ontology*	optional, nullable	Self-reported location of birth of the subject, preferred granularity is country-level
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
genotype	*SubjectGenotype*	optional, nullable	

Diagnosis Fields

Download as TSV

Name	Type	Attributes	Definition
study_group_descr	string	required, nullable	Designation of study arm to which the subject is assigned to
diagnosis_timepoi	*TimePoint*	optional, nullable	Time point for the diagnosis
disease_diagnosis	*Ontology*	required, nullable	Diagnosis of subject
disease_length	*TimeQuantity*	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, identifier, nullable	Sample ID assigned by submitter, unique within study. If possible, a persistent sample ID linked to INSDC or similar repository study should be used.
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_sam	string	required, nullable	Histopathologic evaluation of the sample
collection_time_p	*TimePoint*	required, nullable	Time point at which sample was taken, relative to *label* event
collection_time_p	*Ontology*	DEPRECATED	
collection_time_p	string	DEPRECATED	Event in the study schedule to which *Sample collection time* relates to
collection_locati	*Ontology*	optional, nullable	Location where the sample was taken, preferred granularity is country-level
biomaterial_provi	string	required, nullable	Name and address of the entity providing the sample

Sample Processing Fields

Download as TSV

Name	Type	Attributes	Definition
sample_processing	string	optional, identifier, nullable	Identifier for the sample processing object. This field should be unique within the repertoire. This field can be used to uniquely identify the combination of sample, cell processing, nucleic acid processing and sequencing run information for the repertoire.

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_label	string	optional, nullable	Free text cell type annotation. Primarily used for annotating cell types that are not provided in the Cell Ontology.
cell_species	*Ontology*	optional, nullable	Binomial designation of the species from which the analyzed cells originate. Typically, this value should be identical to *species*, in which case it SHOULD NOT be set explicitly. However, 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_reactic	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_p	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	PhysicalQuantity	required, nullable	Amount of template that went into the process
template_amount_u	*Ontology*	DEPRECATED	
library_generatic	string	required	Generic type of library generation
library_generatic	string	required, nullable	Description of processes applied to substrate to obtain a library that is ready for sequencing
library_generatic	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_sequence	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_prime	string	required, nullable	Position of the most distal nucleotide templated by the forward primer or primer mix
reverse_pcr_prime	string	required, nullable	Position of the most proximal nucleotide templated by the reverse primer or primer mix

Sequencing Data Fields

Download as TSV

Name	Type	Attributes	Definition
sequencing_data_i	string	required, identifier, nullable	Persistent identifier of raw data stored in an archive (e.g. INSDC run ID). Data archive should be identified in the CURIE prefix.
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_direc	string	required, nullable	Read direction for the second file in paired-read sequencing
paired_read_lengt	integer	required, nullable	Read length in bases for the second file in paired-read sequencing
index_filename	string	optional, nullable	File name for the index file
index_length	integer	optional, nullable	Read length in bases for the index file

Sequencing Run Fields

Download as TSV

Name	Type	Attributes	Definition
sequencing_run_id	string	required, identifier, nullable	ID of sequencing run assigned by the sequencing facility
total_reads_passi	integer	required, nullable	Number of usable reads for analysis
sequencing_platfo	string	required, nullable	Designation of sequencing instrument used
sequencing_facili	string	required, nullable	Name and address of sequencing facility
sequencing_run_da	string	required, nullable	Date of sequencing run
sequencing_kit	string	required, nullable	Name, manufacturer, order and lot numbers of sequencing kit
sequencing_files	*SequencingData*	optional	Set of sequencing files produced by the sequencing run

Data Processing Fields

Download as TSV

Name	Type	Attributes	Definition
data_processing_i	string	optional, identifier, nullable	Identifier for the data processing object.
primary_annotatic	boolean	optional, identifier	If true, indicates this is the primary or default data processing for the repertoire and its rearrangements. 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_asse	string	required, nullable	How paired end reads were assembled into a single receptor sequence
quality_threshold	string	required, nullable	How/if sequences were removed from (4) based on base quality scores
primer_match_cutc	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_p	string	required, nullable	General description of how QC is performed
data_processing_f	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.
germline_set_ref	string	optional, nullable	Unique identifier of the germline set and version, in CURIE format
analysis_provenan	string	optional, nullable	Identifier for machine-readable PROV model of analysis provenance

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_label	string	optional, nullable	Free text cell type annotation. Primarily used for annotating cell types that are not provided in the Cell Ontology.
cell_species	*Ontology*	optional, nullable	Binomial designation of the species from which the analyzed cells originate. Typically, this value should be identical to *species*, in which case it SHOULD NOT be set explicitly. However, 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_reactic	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_p	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.

Contributor Fields

Download as TSV

Name	Type	Attributes	Definition
contributor_id	string	required, identifier, nullable	Unique identifier of this contributor within the file
name	string	required	Full name of contributor
orcid_id	string	optional, nullable	ORCID identifier of the contributor. Note that if present, the label of the ORCID record should take precedence over the name reported in the *name* property.
affiliation	string	optional, nullable	ROR of the contributor's primary affiliation. Note that ROR are only minted for institutions, not from individuals institutes, divisions or departments.
affiliation_depar	string	optional, nullable	Additional information regarding the contributor's primary affiliation. Can be used to specify individual institutes, divisions or departments.
contributions	array of Contributor-Contribution	optional, nullable	List of all roles the contributor had in a project

Subject Genotype Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_genotype	*GenotypeSet*	optional, nullable	Immune receptor genotype set for this subject.
mhc_genotype_set	*MHCGenotypeSet*	optional, nullable	MHC genotype set for this subject.

Genotype Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_genotype	string	required, identifier, nullable	A unique identifier within the file for this Receptor Genotype, typically generated by the repository hosting the schema, for example from the underlying ID of the database record.
locus	string	required	Gene locus
documented_allele	array of DocumentedAllele	optional, nullable	List of alleles documented in reference set(s)
undocumented_alle	array of UndocumentedAllele	optional, nullable	List of alleles inferred to be present and not documented in an identified GermlineSet
deleted_genes	array of DeletedGene	optional, nullable	Array of genes identified as being deleted in this genotype
inference_process	string	optional, nullable	Information on how the genotype was acquired. Controlled vocabulary.

Genotype Set Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_genotype	string	required, identifier, nullable	A unique identifier for this Receptor Genotype Set, typically generated by the repository hosting the schema, for example from the underlying ID of the database record.
genotype_class_li	array of *Genotype*	optional, nullable	List of Genotypes included in this Receptor Genotype Set.

MHC Genotype Fields

Download as TSV

Name	Type	Attributes	Definition
mhc_genotype_id	string	required, identifier, nullable	A unique identifier for this MHCGenotype, assumed to be unique in the context of the study
mhc_class	string	required	Class of MHC alleles described by the MHCGenotype
mhc_alleles	array of MHCAllele	required, nullable	List of MHC alleles of the indicated mhc_class identified in an individual
mhc_genotyping_me	string	optional, nullable	Information on how the genotype was determined. The content of this field should come from a list of recommended terms provided in the AIRR Schema documentation.

MHC Genotype Set Fields

Download as TSV

Name	Type	Attributes	Definition
mhc_genotype_set_	string	required, identifier, nullable	A unique identifier for this MHCGenotypeSet
mhc_genotype_list	array of *MHCGenotype*	required, nullable	List of MHCGenotypes included in this set

Time Point Fields

Download as TSV

Name	Type	Attributes	Definition
time_label	string	optional, nullable	Informative label for the time point
time_value	number	optional	Value of the time point
time_unit	*Ontology*	optional	Unit of the time point

Time Interval Fields

Download as TSV

Name	Type	Attributes	Definition
time_min	number	optional	Lower/minimum value of the time interval
time_max	number	optional	Upper/maximum value of the time interval
time_unit	*Ontology*	optional	Unit of the time interval

Time Quantity Fields

Download as TSV

Name	Type	Attributes	Definition
time_quantity	number	optional	Time quantity
time_unit	*Ontology*	optional	Unit of time

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` objects are stored as rows in a *tab-delimited* file and should be compatible with any TSV reader.

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.

Frameshifts

For purposes of annotating alignments, a frameshift is defined as a frameshift that is maintained until the end of the aligned gene, where frames are designated numerically as 1 (in-frame), 2, or 3. For example, an V gene alignment that starts in frame 1 and ends in frame 2, disrupting the conserved cystine, would be defined as a frameshift. Whereas, a V gene alignment with an internal frameshift that corrects with a second frameshift, back to the original frame 1 prior to the conserved cystine, would not need to be annotated as a frameshift.

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 Rearrangement. 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.
quality	string	optional, nullable	The Sanger/Phred quality scores for assessment of sequence quality. Phred quality scores from 0 to 93 are encoded using ASCII 33 to 126 (Used by Illumina from v1.8.)
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.
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.
locus_species	*Ontology*	optional, nullable	Binomial designation of the species from which the locus originates. Typically, this value should be identical to *organism*, if which case it SHOULD NOT be set explicitly. However, there are valid experimental setups in which the two might differ, e.g. transgenic animal models. If set, this key will overwrite the *organism* information for all lower layers of the schema.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
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_alignmen	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.
quality_alignment	string	optional, nullable	Sanger/Phred quality scores for assessment of sequence_alignment quality. Phred quality scores from 0 to 93 are encoded using ASCII 33 to 126 (Used by Illumina from v1.8.)
sequence_alignmen	string	optional, nullable	Amino acid translation of the aligned query sequence.
germline_alignmen	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_alignmen	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.
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.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
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.
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.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
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).
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).

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
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_star	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).
c_sequence_start	integer	optional, nullable	Start position of the C gene in the query sequence (1-based closed interval).
c_sequence_end	integer	optional, nullable	End position of the C gene in the query sequence (1-based closed interval).
c_germline_start	integer	optional, nullable	Alignment start position in the C gene reference sequence (1-based closed interval).
c_germline_end	integer	optional, nullable	Alignment end position in the C gene reference sequence (1-based closed interval).
c_alignment_start	integer	optional, nullable	Start position of the C gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).
c_alignment_end	integer	optional, nullable	End position of the C gene alignment in both the sequence_alignment and germline_alignment fields (1-based closed interval).

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
cdr1_start	integer	optional, nullable	CDR1 start position in the query sequence (1-based closed interval).
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_alignm	string	optional, nullable	Aligned portion of query sequence assigned to the V gene, including any indel corrections or numbering spacers.
v_sequence_alignm	string	optional, nullable	Amino acid translation of the v_sequence_alignment field.
d_sequence_alignm	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_alignm	string	optional, nullable	Amino acid translation of the d_sequence_alignment field.
d2_sequence_align	string	optional, nullable	Aligned portion of query sequence assigned to the second D gene, including any indel corrections or numbering spacers.
d2_sequence_align	string	optional, nullable	Amino acid translation of the d2_sequence_alignment field.
j_sequence_alignm	string	optional, nullable	Aligned portion of query sequence assigned to the J gene, including any indel corrections or numbering spacers.
j_sequence_alignm	string	optional, nullable	Amino acid translation of the j_sequence_alignment field.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
c_sequence_alignm	string	optional, nullable	Aligned portion of query sequence assigned to the constant region, including any indel corrections or numbering spacers.
c_sequence_alignm	string	optional, nullable	Amino acid translation of the c_sequence_alignment field.
v_germline_alignm	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_alignm	string	optional, nullable	Amino acid translation of the v_germline_alignment field.
d_germline_alignm	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_alignm	string	optional, nullable	Amino acid translation of the d_germline_alignment field.
d2_germline_align	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_align	string	optional, nullable	Amino acid translation of the d2_germline_alignment field.
j_germline_alignm	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_alignm	string	optional, nullable	Amino acid translation of the j_germline_alignment field.
c_germline_alignm	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_alignm	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_lengt	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.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
p5d_length	integer	optional, nullable	Number of palindromic nucleotides 5' of the first or only D gene alignment.
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.
v_frameshift	boolean	optional, nullable	True if the V gene in the query nucleotide sequence contains a translational frameshift relative to the frame of the V gene reference sequence.
j_frameshift	boolean	optional, nullable	True if the J gene in the query nucleotide sequence contains a translational frameshift relative to the frame of the J gene reference sequence.
d_frame	integer	optional, nullable	Numerical reading frame (1, 2, 3) of the first or only D gene in the query nucleotide sequence, where frame 1 is relative to the first codon of D gene reference sequence.
d2_frame	integer	optional, nullable	Numerical reading frame (1, 2, 3) of the second D gene in the query nucleotide sequence, where frame 1 is relative to the first codon of D gene reference sequence.
consensus_count	integer	optional, nullable	Number of reads contributing to the UMI consensus or contig assembly 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 identical reads observed for this sequence.
umi_count	integer	optional, nullable	Number of distinct UMIs represented by this sequence. For example, the total number of UMIs that contribute to the contig assembly for the query sequence.
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.
reactivity_id	string	optional, identifier, nullable	Comma separated list of unique identifiers for Reactivity objects associated with this Rearrangement. Order and length of reactivity_id and reactivity_ref are independent.
reactivity_ref	string	optional, nullable	Comma separated list of CURIE identifiers of external reactivity records (e.g. IEDB_EPITOPE:1616345, https://https://www.iedb.org/epitope/1616345) associated with this Rearrangement. Order and length of reactivity_id and reactivity_ref are independent.

continues on next page

Table 2 – continued from previous page

Name	Type	Attributes	Definition
sample_processing	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_i	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_typ	string	optional, nullable	Rearrangement type (source). Most often the type should be "observed," meaning it corresponds to an actual physical sample that was sequenced. Other allowed values are "simulated," i.e. the Rearrangements were generated in silico and there is no linked Subject/Sample metadata, and "inferred" for Rearrangements that are phylogenetically reconstructed from observed sequences. Inferred Rearrangements should be assigned to a Repertoire with repertoire_type=inferred.
rearrangement_id	string	DEPRECATED	Identifier for the Rearrangement object. May be identical to sequence_id, but will usually be a universally unique record locator for database applications.
rearrangement_set	string	DEPRECATED	Identifier for grouping Rearrangement objects.
germline_database	string	DEPRECATED	Source of germline V(D)J genes with version number or date accessed.

Germline Schema

Motivation

Understanding and cataloguing receptor germline genes and allele sequences is critical to the analysis of AIRR data. While the human set is relatively well understood in outline, although probably still far from complete, those of other species, even those that are relatively closely studied, is at a much earlier stage. There is an urgent need to define a standardised format for listing such genes, so that they can be shared between researchers and easily consumed by software tools.

Receptor Germline Schema

The receptor germline schema defines the data elements necessary to describe one or more receptor germline genes, together with supporting evidence. The fundamental object is the `AlleleDescription`, which describes a single gene or allele, containing the necessary details for the annotation of a rearranged sequence such as the location of CDRs (in the case of a V-gene) and framing information (in the case of a J-gene). `AlleleDescription` also contains fields to delineate RSS, and the leader regions of V-genes, should those be covered by the sequence provided.

Evidence supporting the gene or allele can be provided in linked `UnrearrangedSequence` and `RearrangedSequence` objects. Information represented in these objects will typically be stored in a repository: either an INSDC repository such as Genbank or SRA, or a lower-tier repository such as OGRDB. Please note that the key distinction between these object types is whether the V(D)J genes have rearranged, rather than the origin of the material, as mature B and T cells carry rearranged sequences in chromosomal DNA. It is most likely that supporting sequences will be `UnrearrangedSequences`, i.e. prior to rearrangement. In the case of a germline inference from a repertoire, the inferred germline sequence should be provided as a `RearrangedSequence`, if the evidence has been deposited in a

repository.

For V-genes, an IMGT-gapped sequence (i.e.,. a sequence delineated in accordance with the IMGT numbering scheme) is provided in `AlleleDescription`. Other delineations, such as Chothia and Kabat, can be provided via linked `SequenceDelineationV` objects. A `GermlineSet` brings together multiple `AlleleDescriptions` from the same locus to form a curated set. The schema assumes that germline sets will be published by multiple repositories. A germline set may be uniquely referenced by means of the `germline_set_ref`, which is a composite field containing the repository id, germline set label, and version.

Gene and Allele Naming

`AlleleDescription` contains a `label` field, which should contain the accepted name for the field, as determined by the authors/curators of the record. The Nomenclature Committee of the International Union of Immunological Societies (IUIS) allocates gene symbols for receptor genes, and, if a gene symbol has been allocated, this should be used as the label. Where a gene symbol has not been allocated (for example, because the gene or allele has only recently been discovered, or because the available evidence does not meet IUIS standards, a 'temporary label' should be used. It is anticipated that publishers of gene sets will provide mechanisms to issue these temporary labels, and to allow researchers to review change history of `AlleleDescriptions` and `GermlineSets`. To provide consistency across research groups, the Germline Database Working Group of the AIRR Community is developing a community-wide approach to the allocation of temporary labels.

Genotypes

A `GenotypeSet` describes the specific receptor alleles found in a subject, and also identifies genes that are not found (this could be either because they are not present in the chromosomal locus, or because they are not expressed or expressed only at low levels). Depending on the data available and the inference method used, genotypes may contain haplotyping information, which may be full, or partial. As an example of partial haplotyping, the genotype may have been determined from genomic sequencing in which the sequence of the locus was assembled into contigs, but could not be fully assembled. In this case the co-location of alleles in each contig has been established, but the co-location across the entire locus can not be. Co-location is therefore indicated by means of the `phasing` parameter, which in this case would be assigned a different value for alleles on each contig.

MHC Genotypes

Similary to the IG/TR genotypes, the `MHCGenotype` amd `MHCGenotypeSet` objects describe the MHC alleles found in a subject. `MHCGenotype` objects assemble alleles from one class: `MHC-I`, `MHC-II` or `MHC-nonclassical`. The method used to determine the genotype can be provided in the `mhc_genotyping_method` field. As different methods might be use for the various classes, this field is located in the *MHCGenotype* object, not the `MHCGenotypeSet`.

The `mhc_genotyping_method` allows free-text descriptions, however data curators are asked to keep close to the following terms if applicable:

- `PCR-based typing`: Methods whose read-out is the amplification of specific sequences, but which do not provide sequence data by themselves. This includes SSP and SSOP.
- `Sequencing-based typing`: Clinical-grade NGS-based assays, providing high quality and resolution.
- `Inference-based typing`: Allele inferrence based on genome-wide DNA or RNA sequencing.

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`.

Germline Set File Structure

The Germline Set file has a standardised structure that is utilized by all top-level AIRR Schema Objects and defined by the `DataFile` schema. It is intended to contan all information necessary to annotate receptor sequences derived from a single germline locus, and to be directly usable by annotation tools and other processing software.

The file must contain YAML or JSON representation of one or more `GermlineSet` objects, including the associated `AlleleDescription` objects. It may optionally include other associated objects: `SequenceDelineationV`, `RearrangedSequence`, `UnrearrangedSequence`, `Acknowledgement`. These should all be embedded into the overall `GermlineSet` as specified in the schema.

- The file as a whole is considered a dictionary (key/value pair) structure with the keys `Info`, `GermlineSet`, and `AlleleDescription`.
- The `GermlineSet` contains fields `release_version`, `release_description` and `release_date`, which are intended to be used for version identification, under the control of the authors of the `GermlineSet` as identified by the fields `author`, `lab_name` and `lab_address`. If the set is modified by a party other than these authors, that these 6 fields should be modified to reflect the authors of the modification, and their own version identication. These modifications MUST be made if the `GermlineSet` is, or is likely to become, public, in order to avoid confusion with the original set prior to modification. Repositories are encouraged to manage version fields automatically.
- The file can (optionally) contain an `Info` object, at the beginning of the file, based upon the `Info` schema in the OpenAPI 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 `GermlineSet` objects, using `GermlineSet` as the key to the list.
- The file should correspond to a list of `AlleleDescription` objects, using `AlleleDescription` as the key to the list.
- There should be only one `AlleleDescription` for each allele in the list.
- Each `AlleleDescription` object should contain a top-level key/value pair for `allele_description_id` that uniquely identifies the allele description object in the file.
- Each `GermlineSet` object should contain a top-level key/value pair for `germline_set_id` that uniquely identifies the germline set object in the file.
- Some fields require the use of a particular ontology or controlled vocabulary.
- `GermlineSet` and `AlleleDescription` contain reference fields `germline_set_ref` and `allele_description_ref`. These are intended to be globally unique references (containing identifiers of the repository, object and version) that can be used in a query API.
- The structure is the same regardless of whether the data is stored in a file or retrieved from 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.

Schema Field Definitions

GermlineSet Fields

Download as TSV

Name	Type	Attributes	Definition
germline_set_id	string	required, identifier, nullable	Unique identifier of the GermlineSet within this file. Typically, generated by the repository hosting the record.
acknowledgements	array of *Contributor*	required, nullable	List of individuals whose contribution to the germline set should be acknowledged. Note that these are not necessarily identical with the authors on an associated manuscript or other scholarly communication. Further note that typically at least the three CRediT contributor roles "supervision", "investigation" and "data curation" should be assigned. The coresponding author should be listed last.
release_version	integer	required, nullable	Version number of this record, allocated automatically
release_descripti	string	required, nullable	Brief descriptive notes of the reason for this release and the changes embodied
release_date	string	required, nullable	Date of this release
germline_set_name	string	required, nullable	descriptive name of this germline set
germline_set_ref	string	required, nullable	Unique identifier of the germline set and version, in CURIE format
pub_ids	array of string	optional, nullable	Publications describing the germline set
species	*Ontology*	required	Binomial designation of subject's species
species_subgroup	string	optional, nullable	Race, strain or other species subgroup to which this subject belongs
species_subgroup_	string	optional, nullable	
locus	string	required	Gene locus
allele_descriptic	array of *AlleleDescription*	required, nullable	list of allele_descriptions in the germline set
curation	string	optional, nullable	Curational notes on the GermlineSet. This can be used to give more extensive notes on the decisions taken than are provided in the release_description.

AlleleDescription Fields

Download as TSV

Name	Type	Attributes	Definition
allele_descriptic	string	required, identifier, nullable	Unique identifier of this AlleleDescription within the file. Typically, generated by the repository hosting the record.
allele_descriptic	string	optional, nullable	Unique reference to the allele description, in standardized form (Repo:Label:Version)

continues on next page

Table 3 – continued from previous page

Name	Type	Attributes	Definition
acknowledgements	array of *Contributor*	required, nullable	List of individuals whose contribution to the gene description should be acknowledged. Note that these are not necessarily identical with the authors on an associated manuscript or other scholarly communication. Further note that typically at least the three CRediT contributor roles "supervision", "investigation" and "data curation" should be assigned. The current maintainer should be listed first.
release_version	integer	required, nullable	Version number of this record, updated whenever a revised version is published or released
release_date	string	required, nullable	Date of this release
release_descripti	string	required, nullable	Brief descriptive notes of the reason for this release and the changes embodied
label	string	optional, nullable	The accepted name for this gene or allele following the relevant nomenclature. The value in this field should correspond to values in acceptable name fields of other schemas, such as v_call, d_call, and j_call fields.
sequence	string	required	Nucleotide sequence of the gene. This should cover the full length that is available, including where possible RSS, and 5' UTR and lead-in for V-gene sequences.
coding_sequence	string	required, nullable	Nucleotide sequence of the core coding region, such as the coding region of a D-, J- or C- gene or the coding region of a V-gene excluding the leader.
aliases	array of string	optional, nullable	Alternative names for this sequence
locus	string	required	Gene locus
chromosome	integer	optional, nullable	chromosome on which the gene is located
sequence_type	string	required	Sequence type (V, D, J, C)
functional	boolean	required, nullable	True if the gene is functional, false if it is a pseudogene
inference_type	string	required, nullable	Type of inference(s) from which this gene sequence was inferred
species	*Ontology*	required	Binomial designation of subject's species
species_subgroup	string	optional, nullable	Race, strain or other species subgroup to which this subject belongs
species_subgroup_	string	optional, nullable	
status	string	optional, nullable	Status of record, assumed active if the field is not present
subgroup_designat	string	optional, nullable	Identifier of the gene subgroup or clade, as (and if) defined
gene_designation	string	optional, nullable	Gene number or other identifier, as (and if) defined
allele_designatic	string	optional, nullable	Allele number or other identifier, as (and if) defined
allele_similarity	string	optional, nullable	ID of the similarity cluster used in this germline set, if designated

continues on next page

Table 3 – continued from previous page

Name	Type	Attributes	Definition
allele_similarity	string	optional, nullable	Membership ID of the allele within the similarity cluster, if a cluster is designated
j_codon_frame	integer	optional, nullable	Codon position of the first nucleotide in the 'coding_sequence' field. Mandatory for J genes. Not used for V or D genes. '1' means the sequence is in-frame, '2' means that the first bp is missing from the first codon, and '3' means that the first 2 bp are missing.
gene_start	integer	optional, nullable	Co-ordinate in the sequence field of the first nucleotide in the coding_sequence field.
gene_end	integer	optional, nullable	Co-ordinate in the sequence field of the last gene-coding nucleotide in the coding_sequence field.
utr_5_prime_start	integer	optional, nullable	Start co-ordinate in the sequence field of the 5 prime UTR (V-genes only).
utr_5_prime_end	integer	optional, nullable	End co-ordinate in the sequence field of the 5 prime UTR (V-genes only).
leader_1_start	integer	optional, nullable	Start co-ordinate in the sequence field of L-PART1 (V-genes only).
leader_1_end	integer	optional, nullable	End co-ordinate in the sequence field of L-PART1 (V-genes only).
leader_2_start	integer	optional, nullable	Start co-ordinate in the sequence field of L-PART2 (V-genes only).
leader_2_end	integer	optional, nullable	End co-ordinate in the sequence field of L-PART2 (V-genes only).
v_rs_start	integer	optional, nullable	Start co-ordinate in the sequence field of the V recombination site (V-genes only).
v_rs_end	integer	optional, nullable	End co-ordinate in the sequence field of the V recombination site (V-genes only).
d_rs_3_prime_star	integer	optional, nullable	Start co-ordinate in the sequence field of the 3 prime D recombination site (D-genes only).
d_rs_3_prime_end	integer	optional, nullable	End co-ordinate in the sequence field of the 3 prime D recombination site (D-genes only).
d_rs_5_prime_star	integer	optional, nullable	Start co-ordinate in the sequence field of the 5 prime D recombination site (D-genes only).
d_rs_5_prime_end	integer	optional, nullable	End co-ordinate in the sequence field of 5 the prime D recombination site (D-genes only).
j_cdr3_end	integer	optional, nullable	In the case of a J-gene, the co-ordinate in the sequence field of the first nucelotide of the conserved PHE or TRP (IMGT codon position 118).
j_rs_start	integer	optional, nullable	Start co-ordinate in the sequence field of J recombination site (J-genes only).
j_rs_end	integer	optional, nullable	End co-ordinate in the sequence field of J recombination site (J-genes only).
j_donor_splice	integer	optional, nullable	Co-ordinate in the sequence field of the final 3' nucleotide of the J-REGION (J-genes only).
v_gene_delineatic	array of *SequenceDelineationV*	optional, nullable	
unrearranged_supp	array of *UnrearrangedSequence*	optional, nullable	

continues on next page

Table 3 – continued from previous page

Name	Type	Attributes	Definition
rearranged_suppor	array of *Rearranged-Sequence*	optional, nullable	
paralogs	array of string	optional, nullable	Gene symbols of any paralogs
curation	string	optional, nullable	Curational notes on the AlleleDescription. This can be used to give more extensive notes on the decisions taken than are provided in the release_description.
curational_tags	array of string	optional, nullable	Controlled-vocabulary tags applied to this description

RearrangedSequence Fields

Download as TSV

Name	Type	Attributes	Definition
sequence_id	string	required, identifier, nullable	Unique identifier of this RearrangedSequence within the file, typically generated by the repository hosting the schema, for example from the underlying ID of the database record.
sequence	string	required	nucleotide sequence
derivation	string	required, nullable	The class of nucleic acid that was used as primary starting material
observation_type	string	required	The type of observation from which this sequence was drawn, such as direct sequencing or inference from repertoire sequencing data.
curation	string	optional, nullable	Curational notes on the sequence
repository_name	string	required, nullable	Name of the repository in which the sequence has been deposited
repository_ref	string	required, nullable	Queryable id or accession number of the sequence published by the repository
deposited_version	string	required, nullable	Version number of the sequence within the repository
sequence_start	integer	optional	Start co-ordinate of the sequence detailed in this record, within the sequence deposited
sequence_end	integer	optional	End co-ordinate of the sequence detailed in this record, within the sequence deposited

UnrearrangedSequence Fields

Download as TSV

Name	Type	Attributes	Definition
sequence_id	string	required, identifier, nullable	unique identifier of this UnrearrangedSequence within the file
sequence	string	required	Sequence of interest described in this record. Typically, this will include gene and promoter region.
curation	string	optional, nullable	Curational notes on the sequence
repository_name	string	required, nullable	Name of the repository in which the assembly or contig is deposited
repository_ref	string	required, nullable	Queryable id or accession number of the sequence published by the repository
patch_no	string	optional, nullable	Genome assembly patch number in which this gene was determined
gff_seqid	string	required, nullable	Sequence (from the assembly) of a window including the gene and preferably also the promoter region.
gff_start	integer	required, nullable	Genomic co-ordinates of the start of the sequence of interest described in this record in Ensemble GFF version 3.
gff_end	integer	required, nullable	Genomic co-ordinates of the end of the sequence of interest described in this record in Ensemble GFF version 3.
strand	string	required, nullable	sense (+ or -)

SequenceDelineationV Fields

Download as TSV

Name	Type	Attributes	Definition
sequence_delineat	string	required, identifier, nullable	Unique identifier of this SequenceDelineationV within the file. Typically, generated by the repository hosting the record.
delineation_schem	string	required, nullable	Name of the delineation scheme
unaligned_sequenc	string	optional, nullable	entire V-sequence covered by this delineation
aligned_sequence	string	optional, nullable	Aligned sequence if this delineation provides an alignment. An aligned sequence should always be provided for IMGT delineations.
fwr1_start	integer	required, nullable	FWR1 start co-ordinate in the 'unaligned sequence' field
fwr1_end	integer	required, nullable	FWR1 end co-ordinate in the 'unaligned sequence' field
cdr1_start	integer	required, nullable	CDR1 start co-ordinate in the 'unaligned sequence' field
cdr1_end	integer	required, nullable	CDR1 end co-ordinate in the 'unaligned sequence' field
fwr2_start	integer	required, nullable	FWR2 start co-ordinate in the 'unaligned sequence' field
fwr2_end	integer	required, nullable	FWR2 end co-ordinate in the 'unaligned sequence' field
cdr2_start	integer	required, nullable	CDR2 start co-ordinate in the 'unaligned sequence' field
cdr2_end	integer	required, nullable	CDR2 end co-ordinate in the 'unaligned sequence' field
fwr3_start	integer	required, nullable	FWR3 start co-ordinate in the 'unaligned sequence' field
fwr3_end	integer	required, nullable	FWR3 end co-ordinate in the 'unaligned sequence' field
cdr3_start	integer	required, nullable	CDR3 start co-ordinate in the 'unaligned sequence' field
alignment_labels	array of string	optional, nullable	One string for each codon in the aligned_sequence indicating the label of that codon according to the numbering of the delineation scheme if it provides one.

GenotypeSet Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_genotype	string	required, identifier, nullable	A unique identifier for this Receptor Genotype Set, typically generated by the repository hosting the schema, for example from the underlying ID of the database record.
genotype_class_li	array of *Genotype*	optional, nullable	List of Genotypes included in this Receptor Genotype Set.

Genotype Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_genotype	string	required, identifier, nullable	A unique identifier within the file for this Receptor Genotype, typically generated by the repository hosting the schema, for example from the underlying ID of the database record.
locus	string	required	Gene locus
documented_allele	array of DocumentedAllele	optional, nullable	List of alleles documented in reference set(s)
undocumented_alle	array of UndocumentedAllele	optional, nullable	List of alleles inferred to be present and not documented in an identified GermlineSet
deleted_genes	array of DeletedGene	optional, nullable	Array of genes identified as being deleted in this genotype
inference_process	string	optional, nullable	Information on how the genotype was acquired. Controlled vocabulary.

MHCGenotypeSet Fields

Download as TSV

Name	Type	Attributes	Definition
mhc_genotype_set_	string	required, identifier, nullable	A unique identifier for this MHCGenotypeSet
mhc_genotype_list	array of *MHCGenotype*	required, nullable	List of MHCGenotypes included in this set

MHCGenotype Fields

Download as TSV

Name	Type	Attributes	Definition
mhc_genotype_id	string	required, identifier, nullable	A unique identifier for this MHCGenotype, assumed to be unique in the context of the study
mhc_class	string	required	Class of MHC alleles described by the MHCGenotype
mhc_alleles	array of MHCAllele	required, nullable	List of MHC alleles of the indicated mhc_class identified in an individual
mhc_genotyping_me	string	optional, nullable	Information on how the genotype was determined. The content of this field should come from a list of recommended terms provided in the AIRR Schema documentation.

Clone Schema

A `Clone` object groups a set of `Rearrangements` or `Cells` that are inferred to be related by common descent from a single naive ancestor. The member `Rearrangements` and `Cells` are referenced from `Clone` as an array of `Nodes` and can include inferred ancestors that were not directly observed. All members of a `Clone` must be from either a single `RepertoireGroup` or from a single `Repertoire` if a `RepertoireGroup` was not created.

A `Node` links members of a `Clone` to their original metadata and annotations.

File Format Specification

Files are YAML/JSON with an AIRR Data File structure. Files should be encoded as UTF-8. Identifiers are case-sensitive. Files should have the extension `.yaml`, `.yml`, or `.json`.

File Structure

- The *DataFile* is a dictionary (key/value pair) structure with the keys `Clone` and `Node`.
- 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 `Clone` objects, using `Clone` as the key to the list.
- The file should correspond to a list of `Node` objects, using `Node` as the key to the list.
- Each `Clone` object should contain a top-level key/value pair for `clone_id` that uniquely identifies the clone and a top-level key/value pair for *either* `repertoire_group_id` *or* `repertoire_id` that identifies the source of the clone's members.
- Each `Node` object should contain top-level key/value pairs for `node_id` (uniquely identifies the node) and `repertoire_id` (identifies the source 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.

Schema Field Definitions

Clone Fields

`Download as TSV`

Name	Type	Attributes	Definition
clone_id	string	required, identifier	Identifier for the clone.
repertoire_group_	string	required, nullable	Identifier of the RepertoireGroup that this Clone is derived from. If the Clone is derived from a single Repertoire, repertoire_id may be used instead.
repertoire_id	string	required, nullable	Identifier of the Repertoire that this Clone is derived from. Should only be used if Clones are calculated from a single Repertoire without defining a RepertoireGroup. Otherwise use repertoire_group_id instead.
clone_class	string	optional, nullable	Is this a single-chain clone or a cell-based clone?
data_processing_i	string	optional, nullable	Identifier of the data processing object in the repertoire metadata for this clone.
nodes	array of *Node*	required, nullable	List of Nodes that are members of this clone.
inferred_ancestor	string	optional, nullable	Node_id string that acts as a key to the Node record for the inferred naive Rearrangement or Cell for this clone.
clone_count	integer	optional, nullable	Absolute count of the size (number of members) of this clone in the repertoire. This could simply be the number of sequences (Rearrangement records) observed in this clone, the number of distinct cell barcodes (unique cell_id values), or a more sophisticated calculation appropriate to the experimental protocol. Absolute count is provided versus a frequency so that downstream analysis tools can perform their own normalization.
seed_id	string	optional, nullable	sequence_id or cell_id of the seed sequence/cell. Empty string (or null) if there is no seed sequence.
tree	string	optional, nullable	Newick string describing the tree.

Node Fields

Download as TSV

Name	Type	Attributes	Definition
node_id	string	required, identifier	Identifier for the node.
repertoire_id	string	required, nullable	Identifier of the repertoire that cell or rearrangement contained by this node.
cell_id	string	optional, nullable	Identifier of the cell contained by this node. NOTE: Mutually exclusive with Node.sequence_id
sequence_id	string	optional, nullable	Identifier of the rearrangement contained by this node. NOTE: Mutually exclusive with Node.cell_id
node_type	string	optional, nullable	Node type (source). "Observed" for rearrangements/cells that were observed in the source data, "inferred" for unobserved intermediates that were phylogenetically inferred, or "simulated" for simulated rearrangements/cells.
node_class	string	optional, nullable	Does this node contain a rearrangement of a cell?

Single-cell Schema

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

Files are YAML/JSON with an AIRR Data File structure. Files should be encoded as UTF-8. Identifiers are case-sensitive. Files should have the extension `.yaml`, `.yml`, or `.json`.

Schema Field Definitions

Cell Fields

`Download as TSV`

Name	Type	Attributes	Definition
`cell_id`	string	required, identifier	Identifier for the Cell object. This identifier must be unique within a given study, but it is recommended that it be a universally unique record locator to enable database applications.
`repertoire_id`	string	required, nullable	Identifier to the associated repertoire in study metadata.
`data_processing_i`	string	optional, nullable	Identifier of the data processing object in the repertoire metadata for this cell.
`receptors`	array of string	optional, nullable	Array of receptor identifiers defined for the Receptor objects associated with this cell
`cell_subset`	*Ontology*	optional, nullable	Commonly-used designation of isolated cell population.
`cell_phenotype`	string	optional, nullable	List of cellular markers and their expression levels used to isolate the cell population.
`cell_label`	string	optional, nullable	Free text cell type annotation. Primarily used for annotating cell types that are not provided in the Cell Ontology.
`virtual_pairing`	boolean	required, nullable	boolean to indicate if pairing was inferred.
`cell_type`	string	optional, nullable	Cell type (source). Most often the type should be "observed," meaning it corresponds to an actual physical sample that was sequenced. Other allowed values are "simulated," i.e. the Cells were generated in silico and there is no linked Subject/Sample metadata, and "inferred" for Cells that are phylogenetically reconstructed from observed sequences. Inferred Cells should be assigned to a Repertoire with repertoire_type=inferred.

Expression Fields

`Download as TSV`

Name	Type	Attributes	Definition
expression_id	string	required, identifier	Identifier for the Expression object. This identifier must be unique within a given study, but it is recommended that it be a universally unique record locator to enable database applications.
cell_id	string	required	Identifier of the cell to which this expression data is related.
repertoire_id	string	required, nullable	Identifier for the associated repertoire in study metadata.
data_processing_i	string	required, nullable	Identifier of the data processing object in the repertoire metadata for this cell.
property_type	string	required	Keyword describing the property type and detection method used to measure the property value. The following keywords are recommended, but custom property types are also valid: "mrna_expression_by_read_count", "protein_expression_by_fluorescence_intensity", "antigen_bait_binding_by_fluorescence_intensity", "protein_expression_by_dna_barcode_count" and "antigen_bait_binding_by_dna_barcode_count".
property	string	required, nullable	Name of the property observed, typically a gene or antibody identifier (and label) from a canonical resource such as Ensembl (e.g. ENSG00000275747, IGHV3-79) or Antibody Registry (ABREG:1236456, Purified anti-mouse/rat/human CD27 antibody).
value	number	required, nullable	Level at which the property was observed in the experiment (non-normalized).

Reactivity & Receptor Schemas

Reactivity Schema

The `Reactivity` object contains information that describes the binding of a compound resembling an Ig or TCR antigen by a single, intact cell. It is critical to note that while such experimental measurements are related to the antigen reactivity of individual Receptors expressed by the cell, the relation is rather complex as multiple Receptor species, different expression levels and background binding of the compound would need to be taken into account. Therefore the AIRR Schema provides a separate record for this information, which is only indirectly linked (via `Cell`) to the `Receptor` object.

Receptor Schema

The purpose of the `Receptor` object is to provide an structure for information referring to actual *Receptors*, i.e., Ig or TCR, both for outgoing and incoming links. To this end, the `Receptor` object describes the receptor as an abstract and global concept, i.e., the actual Ig/TCR protein complex, which MAY or MAY NOT have been observed in the current study. However, the rearrangements encoding the respective chains MUST be present in the study as well as the information linking them (see below). In addition the object allow references to entries in external database (e.g., IEDB).

The `Receptor` object explicitly requires full sequence information of the two associated variable domains. This is considered to be an acceptable restriction from an AIRR-seq perspective, where sequencing typically precedes or takes place in combination with the determination of receptor reactivity.

Identifiers

The `Receptor` objects has two properties that serve as identifiers:

- `receptor_id` is a **local** identifier and its uniqueness MUST NOT be assumed beyond the scope of the study the receptor was reported in. This property can be used, e.g., to represent designations for Ig/TCR used in a manuscript.
- `receptor_hash` is the SHA256 hash of the receptors variable domain amino acid sequences, which serves as a **globally unique** identifier that can be independently calculated by repositories without requiring prior communication. It is calculated as follows, where `base16` designates the function described in RFC4648 Section 6:

```
lower_case(
    base16(
        sha256(
            concatenate(
                upper_case(receptor_variable_domain_1_aa),
                upper_case(receptor_variable_domain_2_aa)
            )
        )
    )
)
```

receptor_variable_domain_1_aa is the complete amino acid sequence of the mature variable domain of the Ig heavy, TCR beta or TCR delta chain. *receptor_variable_domain_2_aa* is the complete amino acid sequence of the mature variable domain of the Ig light, TCR alpha or TCR gamma chain.

Relations to other AIRR Schema objects

The `Receptor` object is only directly linked to the `Cell` object, which then in turn contains the references to the records in the `Rearrangements` that encode the respective chains of the receptor. Therefore a given rearrangement cannot directly reference to a receptor, which is also not a meaningful thing to do, as the paired chain would be unclear, but is necessary to determine a receptors reactivity.

Annotation guidelines

References to information describing the same receptor located in other databases (i.e., outgoing links) SHOULD be provided as as CURIEs in the `receptor_ref` property. Entries in this array MUST refer to objects that a conceptually similar to the *Receptor* concept used by the AIRR Schema. Linkage to potentially existing reactivity information needs is expected to happen in the external database, not in the `Receptor` record.

`Receptor` objects SHOULD be created even in the absence of additional external information, as this will enhance the discoverability of AIRR-seq experiments in which a receptor might have been present. This especially applies to experiments that provide further evidence (e.g., surface expression, reaction to superantigens) showing that a receptor is functional and present on the surface.

Note on cells expressing more than a single receptor

Cells that express more than a single IGH/TRB/TRD or a single IGK/IGL/TRA/TRG chain are regularly observered as allelic exclusion is never complete and its efficiency is rather low for loci like TRA. Such dual-expressing cells can technically be accommodated in the current AIRR Schema as an individual `Cell` object can link to more than two rearrangements and to more than a single `Receptor`. In the case of two potential receptors, both MAY be created as objects, if the general annotation rules are met for each of them. Note that the annotation of cell-based reactivity information is handled by the *Reactivity & Receptor Schemas* object.

Representation of bi-specific antibodies

The goal of the AIRR Standards is primarily to represent naturally occuring receptors. While bi-specific antibodies may arise in dual IGK/IGL expressing B cells their individual reactivity is not measured on a regular basis. Therefore they are currently not supported in the `Receptor` schema.

Schema Field Definitions

Reactivity Fields

`Download as TSV`

Name	Type	Attributes	Definition
reactivity_id	string	required, identifier	Identifier for the Reactivity object. This identifier must be unique within a given study, but it is recommended that it be a universally unique record locator to enable database applications.
cell_id	string	required, nullable	Identifier of the Cell in the context of which the reactivity measurement was conducted.
repertoire_id	string	optional, nullable	Identifier for the associated repertoire in study metadata.
data_processing_i	string	optional, nullable	Identifier of the data processing object in the repertoire metadata for this cell.
ligand_type	string	required	Classification of ligand binding to the cell
antigen_type	string	required	The type of antigen before processing by the immune system.
antigen	string	required	The substance against which the receptor was tested. This can be any substance that stimulates an adaptive immune response in the host, either through antibody production or by T cell activation after presentation via an MHC molecule.
antigen_source_sp	*Ontology*	optional, nullable	The species from which the antigen was isolated
peptide_start	integer	optional, nullable	Start position of the peptide within the reference protein sequence
peptide_end	integer	optional, nullable	End position of the peptide within the reference protein sequence
peptide_sequence_	string	optional, nullable	The actual peptide sequence against which the receptor reactivity was measured. This field should be used as a convenience for antigens of antigen_type *protein* or *peptide*.
mhc_class	string	optional, nullable	Class of MHC molecule, only present for MHC:x ligand types
mhc_gene_1	*Ontology*	optional, nullable	The MHC gene to which the mhc_allele_1 belongs
mhc_allele_1	string	optional, nullable	Allele designation of the MHC alpha chain
mhc_gene_2	*Ontology*	optional, nullable	The MHC gene to which the mhc_allele_2 belongs
mhc_allele_2	string	optional, nullable	Allele designation of the MHC class II beta chain or the invariant beta2-microglobin chain
reactivity_method	string	required	The methodology used to assess/classify reactivity. This should be either the assay utlizied in the experiment, delineated as "annotated" if annotated from an external source (e.g. IEDB), or inferred if imputed using a computational method. In future versions we anticipate this field being an enumerated type, and it is strongly recommended that users utilize one of the following keywords in this field if appropriate: native_protein, MHC_peptide_multimer, annotated, inferred.
reactivity_readou	string	required	Reactivity measurement read-out. For physical assays, this should describe what was measured. For inferred and annotated methods this should indicate a confidence/quality level for the inference/annotation. In future versions we anticipate this field being an enumerated type, and it is strongly recommended that users utilize one of the following keywords in this field if appropriate: fluorescence_intensity, barcode_count, confidence.
reactivity_value	number	required	The absolute (processed) value of the measurement
reactivity_unit	string	required	The unit of the measurement
reactivity_refs	array of string	optional, nul-	Array of cross references to external epitope reactivity

Within the Reactivity object, it is expected that the properties antigen_source_species, peptide_start, peptide_end and peptide_sequence_aa have an inseparable relationship with antigen_type. They only present a valid value when antigen_type is protein or peptide, otherwise they MUST contain a NULL value. In the former case, peptide_sequence_aa SHOULD present the actual peptide sequence of the protein used experimentally, while the antigen field SHOULD reference to a database entry of the protein from which the peptide was derived from. Both peptide_start and peptide_end indicate the (1-based) start and end location of peptide_sequence_aa in the reference sequence. Note that highly- repetitive proteins might contain the same peptide at multiple locations of their full-length sequence. While it is generally recommended to always use the position of the first occurence for the peptide_start and peptide_end annotation, this also stresses the importance to compare actual peptide sequences, not only coordinates.

The five MHC properties (mhc_*), which are specifically required for records in which ligand_type is MHC:peptide or MHC:non-peptide should be NULL for all other ligand_types.

Receptor Fields

Download as TSV

Name	Type	Attributes	Definition
receptor_id	string	required, identifier	Identifier for the Receptor object. This identifier must be unique within a given study, but it is recommended that it be a universally unique record locator to enable database applications.
receptor_hash	string	required	The SHA256 hash of the receptor amino acid sequence, calculated on the concatenated receptor_variable_domain_*_aa sequences and represented as base16-encoded string.
receptor_type	string	required	The top-level receptor type, either Immunoglobulin (Ig) or T Cell Receptor (TCR).
receptor_variable	string	required	Complete amino acid sequence of the mature variable domain of the Ig heavy, TCR beta or TCR delta chain. The mature variable domain is defined as encompassing all AA from and including first AA after the the signal peptide to and including the last AA that is completely encoded by the J gene.
receptor_variable	string	required	Locus from which the variable domain in receptor_variable_domain_1_aa originates
receptor_variable	string	required	Complete amino acid sequence of the mature variable domain of the Ig light, TCR alpha or TCR gamma chain. The mature variable domain is defined as encompassing all AA from and including first AA after the the signal peptide to and including the last AA that is completely encoded by the J gene.
receptor_variable	string	required	Locus from which the variable domain in receptor_variable_domain_2_aa originates
receptor_ref	array of string	optional, nullable	Array of receptor identifiers defined for the Receptor object

Schema Release Notes

Version 2.0.0: June 5, 2026

Version 2.0 major schema release.

General Schema Changes:

- Reorganized time, physical quantities, and contributor tracking by introducing reusable component schemas: `TimeInterval`, `PhysicalQuantity`, `TimeQuantity`, `Contributor`, and `ContributorContribution`.
- Replaced the `Acknowledgement` schema with the more robust `Contributor` schema, which incorporates the CRediT contributor taxonomy roles and ROR institutional identifiers.
- Expanded `CURIEMap` with new prefixes for geographic locations and germline repositories: `GAZ`, `IEDB_EPITOPE`, `IMGT_GERMLINESET`, and `OGRDB_GERMLINESET`.
- Added the `adc-api-optional` attribute to the `Attributes` schema to designate optional API query fields.
- Introduced data package management schemas: `FileObject`, `DataSet`, and `Manifest` to support metadata bundling for groups of files.
- Updated the `DataFile` schema to include arrays for `Node` and `Manifest` records, while removing direct nesting of `CellExpression`.
- Removed the obsolete `Alignment` schema definition.

Time and Quantity Reorganization:

- Simplified the `TimePoint` schema fields by shortening prefixes from `time_point_` to `time_` (e.g., `time_label`, `time_value`, and `time_unit`).
- Converted fields representing single numbers or strings with separate units into unified object references: + `Subject.age` now uses `TimeInterval` (replacing `age_min`,

  `age_max`, and `age_unit`).

  - `Diagnosis.disease_length` now uses `TimeQuantity` (replacing the generic string).
  - `Sample.collection_time_point_relative` now uses a `TimePoint` object reference (replacing separate number, unit, and reference fields).
  - `NucleicAcidProcessing.template_amount` now uses `PhysicalQuantity` (replacing separate number and unit fields).

Repertoire Schema:

- Reorganized study authorship and contact tracking in the `Study` schema by introducing a unified `contributors` array, deprecating individual contact fields (`study_contact`, `collected_by`, `lab_name`, `lab_address`, and `submitted_by`).
- Changed the type of `Study.pub_ids` from a single string to an array of strings.
- Added a `repertoire_type` field to the `Repertoire` schema supporting a controlled vocabulary (`observed`, `simulated`, `inferred`, `null`).
- Added an optional `filter` object to the `RepertoireFilter` schema to document how rearrangements or cells were filtered using JSON structures consistent with the ADC API.
- Enhanced geographic and demographic tracking by converting `Subject.ancestry_population` to an `Ontology` reference and adding `Subject.location_birth` and `Sample.collection_location` as new ontology fields.

Rearrangement Schema:

- Added the `locus_species` ontology field to support transgenic or chimeric models where the locus species differs from the host organism.
- Added a `rearrangement_type` field supporting a controlled vocabulary (`observed`, `simulated`, `inferred`, `null`).
- Added `reactivity_id` and `reactivity_ref` fields to link rearrangement records directly with single-cell reactivity data.
- Formalized the deprecation of `rearrangement_id` (merged with `sequence_id`), `rearrangement_set_id` (replaced by specific identifiers), and `germline_database` (moved to `DataProcessing`).

Single-cell Schema:

- Renamed the `CellExpression` schema to `Expression` and its `property_value` field to `value` for clarity and brevity.
- Renamed and overhauled the `ReceptorReactivity` schema into a top-level single-cell data object called `Reactivity`, adding keys like `reactivity_id`, `cell_id`, `repertoire_id`, and `data_processing_id` to link directly to individual cells and data processing records.
- Removed the `reactivity_measurements` block from the `Receptor` schema, shifting reactivity tracking to the new top-level `Reactivity` object.
- Updated the `Cell` schema by adding `cell_subset`, `cell_phenotype`, `cell_label`, and `cell_type` fields, while removing the legacy rearrangements array and expression metadata fields.
- Added a `cell_label` free text field to the `CellProcessing` schema for custom cell annotations not captured by standard ontologies.

Clone and Tree Schema:

- Completely restructured the `Clone` schema to support multi-repertoire clonal analysis across an entire `RepertoireGroup`.
- Replaced clone-level alignment and sequence annotations (such as `v_call`, `d_call`, `j_call`, `junction`, and individual coordinates) with a required list of `Node` records and an `inferred_ancestor` reference.
- Embedded phylogenetic trees directly within the `Clone` object as a Newick-formatted string field (`tree`), leading to the removal of the separate `Tree` schema.
- Redefined the `Node` schema to serve as a link between a clone member and its original repertoire, specifying its source via mutually exclusive `cell_id` or `sequence_id` fields, along with descriptive properties like `node_type` and `node_class`.

Version 1.6.0: July 7, 2025

Version 1.6 schema release.

- Added `minimum` value to multiple numeric fields.
- Made `TimePoint` fields `label`, `value` and `unit` unique by prefixing with `time_point_`
- Renamed `Acknowledgement.name` to `individual_full_name`.
- Renamed `CellExpression.value` to `property_value`.
- Moved several `RepertoireGroup` fields into a separate `RepertoireFilter` object.
- Fixed multiple type and example errors in the various schemas.

Version 1.5.1: June 2, 2024

Version 1.5 patch release.

- Corrected schema version number in the `Info` object.

Version 1.5.0: August 29, 2023

Version 1.5 schema release.

General Schema Changes:

1. Fixed synchronization errors between the OpenAPI v2 and v3 versions of the AIRR Schema (airr-schema.yaml and airr-schema-openapi3.yaml).
2. Set the default value of `x-airr.miarr` attributes to `defined`.
3. Converted all `x-airr.format` attribute values to snake_case, which specifically impacts any instance of `controlled vocabulary` or `physical quantity`.
4. Corrected numerous instances of missing *x-airr.miairr* and *x-airr.identifier* attributes.
5. Replaced `x-airr.adc-api-optional` attribute with `x-airr.adc-query-support` in multiple fields.
6. Added "IGI" as a valid value to the `locus` enum fields in multiple schema.
7. Added `null` as a valide value to all nullable enum fields.
8. Removed `discriminator: AIRR` from all object definitions.

Germline and Genotype Schema:

1. Clarified the descriptions of multiple fields in the Germline and Genotype schema.
2. Modified `x-airr: nullable` and `x-airr: identifier` values on multiple fields in the Germline and Genotype schema.
3. Removed the `alignment` field and added the `unaligned_sequence`, `aligned_sequences`, and `alignment_labels` fields to the `SequenceDelineationV` object.
4. Converted the enum values in the `inference_type` field of `AlleleDescription` to snake_case.
5. Added the `allele_similarity_cluster_designation` and `allele_similarity_cluster_member_id` fields to `AlleleDescription`.
6. Moved the nested objects `DocumentedAllele`, `UndocumentedAllele`, and `DeletedGenes` out of `Genotype` and defined them as top-level objects references by the `documented_alleles`, `undocumented_alleles`, and `deleted_genes` fields, respectively.
7. Moved the nested object `MHCAllele` out of `MHCGenotype` and defined it as a top-level object referenced by the `mhc_alleles` field.

Single-cell Schema:

1. Added the `property_type` field to the `CellExpression` object.
2. Moved the nested `ReceptorReactivity` object out of `Receptor` and defined it as a top-level object referenced by the `reactivity_measurements` field.

Subject Schema:

1. Removed the nested references to `GenotypeSet` and `MHCGenotypeSet` in the `genotype` field and modified the definition to point to a top-level `SubjectGenotype` object defining these references.

DataProcessing Schema:

1. Clarified the description of `quality_thresholds` to indicate that quality filtering is not mandatory.

Version 1.4.1: August 27, 2022

Version 1.4 schema release.

New General Purpose Schema:

1. Introduced the experimental `DataFile` object, which defines a JSON file holding Repertoire metadata, data processing analysis objects, or any object in the AIRR Data Model.
2. Introduced the experimental `RepertoireGroup` Schema for describing collections of repertoires to be analyzed together.
3. Introduced the experimental `InfoObject` Schema, which provides information about data and ADC API responses.
4. Introduced the experimental `TimePoint` Schema for defining the time point at which an observation or other action was performed.

New Germline and Genotype Schema:

The following experimental schema were introduced to support storage of VDJ germline reference sequences, VDJ genotypes, and MHC genotypes:

1. `GermlineSet`: Defines a collection of `AlleleDescriptions` from the same strain or species.
2. `AlleleDescription`: Details of a putative or confirmed Ig receptor gene/allele inferred from one or more observations.
3. `RearrangedSequence`: Details of a directly observed rearranged sequence or an inference from rearranged sequences contributing support for a gene or allele.
4. `UnrearrangedSequence`: Details of an unrearranged sequence contributing support for a gene or allele.
5. `SequenceDelineationV`: Delineation of a V-gene in a particular system.
6. `GenotypeSet`: Defines a collection a VDJ genotypes for a given subject.
7. `Genotype`: Enumerates the alleles and gene deletions inferred in a single subject for a single locus.
8. `MHCGenotypeSet`: Defines a collection of MHC genotypes for a given subject.
9. `MHCGenotype`: Details the genotype of major histocompatibility complex (MHC) class I, class II and non-classical loci.
10. `Acknowledgement`: Defines contributors to the germline or genotype description.

New Single-cell Schema:

The following experimental schema were introduced to improve support for single-cell data and extend the `Cell` schema.

1. `CellExpression`: Defines a container to store single-cell expression level measurements.
2. `Receptor`: Describes a complete receptor protein sequence and its reactivity.

Rearrangement Schema:

1. Added the optional fields `v_frameshift`, `j_frameshift`, `d_frame` and `d2_frame` defining annotations related to alignment reading frames.
2. Added the optional field `umi_count` to represent the count of distinct UMIs for a sequence.
3. Modified the definition of `duplicate_count` to remove ambiguity with the new `umi_count` field in a single-cell context. There is now a distinction between duplicate observed sequences (`duplicate_count`) and UMIs (`umi_count`).

4. The optional `quality` and `quality_alignment` alignment fields were added to store Phred quality scores for base calls in the `sequence` and `sequence_alignment` fields, respectively.
5. The following optional fields were added to denote constant region (`c_call`) alignment positions: `c_sequence_start`, `c_sequence_end`, `c_germline_start`, `c_germline_end`, `c_alignment_start`, `c_alignment_end`.

Study Schema:

1. Added the optional fields `study_contact` to store contact information for the primary study contact.
2. Modified the enumerated values supported by `keywords_study` to the following set: `contains_ig`, `contains_tr`, `contains_paired_chain`, `contains_schema_rearrangement`, `contains_schema_clone`, `contains_schema_cell`, `contains_schema_receptor`
3. Added the optional fields `adc_publish_date` and `adc_update_data` that timestamp AIRR Data Commons initial publication and last update, respectively.

Subject Schema:

1. Added the optional `genotype` field linking to the new `GenotypeSet` and `MHCGenotypeSet` objects.

Sample Schema:

1. Added the required field `collection_time_point_relative_unit` defining the units for the sample collection timestamp.
2. Modified the type of the field `collection_time_point_relative` from a string to a number defined in combination with the new unit ontology field `collection_time_point_relative_unit`.

NucleicAcidProcessing Schema:

1. Added the required field `template_amount_unit` defining the units for the input template quantification.
2. Modified the type of the `template_amount` field from a string to a number defined in the combination with the new unit ontology field ``template_amount_unit`.

Clone Schema:

1. Added the optional `clone_count` field to specify absolute count of clonal members.
2. Added the optional `umi_count` field to specify the total UMI count of all clonal members.

Cell Schema:

1. Removed the field `expression_tabular` whose functionality has been replaced by the new `CellExpression` schema.

Version 1.3.1: October 13, 2020

Version 1.3 documentation patch release.

Alignment Schema:

1. Added the deprecation tags for `rearrangement_id`, which were accidentally left out of the v1.3.0 release.

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.

2.3.2 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 AIRR 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.
Diagnosi	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).
CellProc	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.
NucleicA	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).
Sequenci	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).
DataProc	Information about the data processing to transform the raw sequencing data into Rearrangements.
Repertoi	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*.
Repertoi	Composite object that combines multiple Repertoires (as RepertoireFilters) for further analysis.
Repertoi	Object with a pointer to an original Repertoire with descriptions of how it was filtered for inclusion in a RepertoireGroup and optional additional metadata such as time point. RepertoireFilters have their own schema and file format described *here*.
Rearrang	Annotated sequences describing adaptive immune receptor chains. Rearrangements have their own schema and file format described *here*.
Clones	Information about inferred clones/lineages from a study. Clones have their own schema and file format described *here*.
Cells	Information about an observed Cell in a study. Cells have their own schema and file format described *here*.
CellExpr properti	Information about expression properties observed for a specific cell. CellExpression properties have their own schema and file format described here.
Receptor	Information about adaptive immune receptors (i.e., Ig and TCR) that are linked to observed Cells in a study. Receptors have their own schema and file format described *here*.
Germline	Lists the receptor germline sequences that have been identified for a single locus within a particular species or sub-species, together with supporting evidence and additional metadata to assist with sequence annotation. Brings togteher the subsidiary objects AlleleDescription, SequenceDelineationV, RearrangedSequence, UnrearrangedSequence, Acknowledgement.
Genotype	Lists the receptor germline sequences that have been identified within a single subject, including both those that are listed within GermlineSets and those that have not been so listed. References the subsidiary object Genotype, which covers a single locus.

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.
- `SampleProcessing` 1-to-1 with `Sample`, `CellProcessing`, `NucleicAcidProcessing`, and `SequencingRun`. A sample processing is a single chain from initial collection, through cell and nucleic acid processing, to sequencing.
- `Repertoire` 1-to-n with `SampleProcessing`. Generally a repertoire has a single sample processing, but sometimes studies perform technical replicates or re-sequencing to generate additional data, and these studies will have multiple sample processings, 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`, `sample_id`, and `sample_processing_id` fields to uniquely identify the `Study`, `Subject`, `Sample`, and `SampleProcessing` objects across multiple repertoires. The exception is that `CellProcessing` and `NucleicAcidProcessing` do not have their own unique identifiers, so they are included within `SampleProcessing`.

As a `Repertoire` is limited to a single sample, many analyses will involve multiple `Repertoires`, which may be combined into a `RepertoireGroup`.

AIRR extension properties

The OpenAPI V2 and V3 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. This extension is not valid for OpenAPI V3 as the `nullable` builtin property should be used.
`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.
`identifier`	True if the field is an identifier required to link metadata and/or individual sequence records across objects in the complete AIRR Data Model and ADC API.
`adc-query-su`	True if an ADC API implementation must support queries on the field. If false, query support for the field in ADC API implementations is optional.
`adc-api-opt:`	True if the field is specific to the ADC API and is not part of the AIRR specification proper. These are typically "convenience" fields that make finding data easy or efficient (can be optimized by a repository).
`deprecated`	True if the field has been deprecated from the schema.
`deprecated-(`	Information regarding the deprecation of the field.
`deprecated-]`	The deprecated field is replaced by this list of fields.

2.3.3 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.

2.4 Data Submission and Query

2.4.1 Submission of Processed AIRR-seq Studies

There are multiple data repositories that accept submission of processed (analysis ready) AIRR-seq datasets. Each provides different capabilities, but all comply with the AIRR Data Standards.

VDJServer Community Data Portal
iReceptor Turnkey Repository

2.4.2 Raw Sequence Data Submission

The NCBI Sequence Read Archive (SRA) is recommended for raw sequence (FASTQ) data submission with BioProject and BioSample metadata conforming to the *MiAIRR requirements*.

2.4.3 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.

Open Germline Receptor Database

2.4.4 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.

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 Standards Reference Implementations

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

AIRR Data Model Interface

airr.**read_airr**(*filename*, *format=None*, *validate=False*, *model=True*, *debug=False*, *check_nullable=True*)

Load an AIRR Data file

Parameters

- **filename** (*str*) – Path to the input file.
- **format** (*str*) – Input file format valid strings are "yaml" or "json". If set to None, the file format will be automatically detected from the file extension.
- **validate** (*bool*) – Whether to validate data as it is read, raising a ValidationError exception in the event of a validation failure.
- **model** (*bool*) – If True only validate objects defined in the AIRR DataFile schema. If False, attempt validation of all top-level objects. Ignored if validate=False.
- **debug** (*bool*) – Debug flag. If True print debugging information to standard error.
- **check_nullable** (*bool*) – Whether to check for nullable fields when validating the data.

Returns
Dictionary of AIRR Data objects.

Return type
dict

airr.**write_airr**(*filename*, *data*, *format=None*, *info=None*, *validate=False*, *model=True*, *debug=False*, *check_nullable=True*)

Write an AIRR Data file

Parameters

- **filename** (*str*) – Path to the output file.
- **data** (*dict*) – Dictionary of AIRR Data Model objects.
- **format** (*str*) – Output file format valid strings are "yaml" or "json". If set to None, the file format will be automatically detected from the file extension.
- **info** (*object*) – Info object to write. Will write current AIRR Schema info if not specified.
- **validate** (*bool*) – Whether to validate data before it is written, raising a ValidationError exception in the event of a validation failure.
- **model** (*bool*) – If True only validate and write objects defined in the AIRR DataFile schema. If False, attempt validation and write of all top-level objects.
- **debug** (*bool*) – Debug flag. If True print debugging information to standard error.
- **check_nullable** (*bool*) – Whether to check for nullable fields if validating the data.

Returns
True if the file is written without error.

Return type
bool

airr.**validate_airr**(*data*, *model=True*, *debug=False*, *check_nullable=True*)

Validates an AIRR Data file

Parameters

- **data** (*dict*) – Dictionary containing AIRR Data Model objects
- **model** (*bool*) – If True only validate objects defined in the AIRR DataFile schema; If False attempt validation of all top-level objects.
- **debug** (*bool*) – Debug flag. If True print debugging information to standard error.
- **check_nullable** (*bool*) – Whether to check for nullable fields when validating the data.

Returns
True if files passed validation; otherwise False.

Return type
bool

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 writer 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

definition

name of the schema definition.

`info`

schema info.

Type
collections.OrderedDict

`properties`

field definitions.

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.

`pandas_types`()

Map of schema types to pandas types

Returns
mapping dictionary for pandas types

Return type
dict

`spec`(*field*)

Get the properties for a field

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

Returns
definition for the field.

Return type
collections.OrderedDict

`template`()

Create an empty template object

Returns
dictionary with all schema properties set as None or an empty list.

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*, *check_nullable=True*)

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
- **check_nullable** (*bool*) – check if data complies with the required fields as determined by the nullable flag.

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.**InfoSchema Schema object for the Info definition**

AIRR schema definitions

airr.schema.**definition**

name of the schema definition.

airr.schema.**info**

schema info.

Type
collections.OrderedDict

airr.schema.**properties**

field definitions.

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.**DataFileSchema Schema object for the DataFile definition**

AIRR schema definitions

airr.schema.**definition**

name of the schema definition.

airr.schema.**info**

schema info.

Type
collections.OrderedDict

airr.schema.**properties**

field definitions.

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.**definition**

name of the schema definition.

airr.schema.**info**

schema info.

Type
collections.OrderedDict

airr.schema.**properties**

field definitions.

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.**definition**
name of the schema definition.

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

Type
collections.OrderedDict

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

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.**GermlineSetSchema Schema object for the Repertoire definition**

AIRR schema definitions

airr.schema.**definition**

name of the schema definition.

airr.schema.**info**

schema info.

Type
collections.OrderedDict

airr.schema.**properties**

field definitions.

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.**GenotypeSetSchema Schema object for the Repertoire definition**

AIRR schema definitions

airr.schema.**definition**

name of the schema definition.

airr.schema.**info**

schema info.

Type
collections.OrderedDict

airr.schema.**properties**

field definitions.

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

Deprecated

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
Dictionary of AIRR Data objects.

Return type
dict

Deprecated since version 1.4: Use *read_airr()* instead.

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

Deprecated since version 1.4: Use *write_airr()* instead.

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

Deprecated since version 1.4: Use *validate_airr()* instead.

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

Deprecated since version 1.4: Use *schema.Schema.template()* instead.

Commandline Tools

Deprecated since version 1.4: The `validate repertoire` subcommand is deprecated. Use `validate airr` instead.

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 files for AIRR Standards compliance.

```
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 airr

Validate AIRR Data Model files.

```
usage: airr-tools validate airr [--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 Data Model files.

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 2.0.0: June 5, 2026

1. Added `check_nullable` argument to `read_airr`, `write_airr`, `validate_airr`, and `Schema.validate_object` to control validation of required and nullable fields.
2. Updated `load_rearrangement` to catch exceptions during file loading, log errors to stderr, and return `None`.
3. Removed `Alignment` schema and `AlignmentSchema`.
4. Renamed `Acknowledgement` schema to `Contributor`.
5. Simplified `derive_rearrangement` by removing explicit gzip file handling.

Version 1.6.1: February 12, 2026

1. Fixed data overwrite bug in `RearrangementWriter`.
2. Fixed pkg_resources import error with setuptools>=82.0.0.

Version 1.6.0: July 7, 2025

Updated schema.

Version 1.5.1: June 2, 2024

1. Updated versioneer to v0.29.

Version 1.5.0: August 29, 2023

1. Updated schema set and examples to v1.5.
2. Officially dropped support for Python 2.
3. Added check for valid enum values to schema validation routines.

4. Set enum values to first defined value during template generation routines.
5. Removed mock dependency installation in ReadTheDocs environments from setup.
6. Improved package import time.

Version 1.4.1: August 27, 2022

General:

1. Updated pandas requirement to 0.24.0 or higher.
2. Added support for missing integer values (`NaN`) in `load_rearrangement` by casting to the pandas `Int64` data type.
3. Added gzip support to `read_rearrangement`.
4. Significant internal refactoring to improve schema generalizability, harmonize behavior between the python and R libraries, and prepare for AIRR Standards v2.0.
5. Fixed a bug in the `validate` subcommand of `airr-tools` causing validation errors to only be reporting for the first invalid file when multiple files were specified on the command line.

Data Model and Schema:

1. Added support for arrays of objects in a single JSON or YAML file.
2. Added support for the AIRR Data File and associated schema (DataFile, Info). The Data File data format holds AIRR object of multiple types and is backwards compatible with Repertoire metadata.
3. Added support for the new germline and genotyping schema (GermlineSet, GenotypeSet) and associated schema.
4. Renamed `schema.CachedSchema` to `schema.AIRRSchema`.
5. Removed `specs/blank.airr.yaml`.

Deprecations:

1. Deprecated `load_repertoire`. Use `read_airr` instead.
2. Deprecated `write_repertoire`. Use `write_airr` instead.
3. Deprecated `validate_repertoire`. Use `validate_airr` instead.
4. Deprecated `repertoire_template`. Use `schema.RepertoireSchema.template` instead.
5. Deprecated the commandline tool `airr-tools validate repertoire`. Use `airr-tools validate airr` instead.

Version 1.3.1: October 13, 2020

1. Refactored `merge_rearrangement` to allow for larger number of files.
2. Improved error handling in format validation operations.

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

Deprecation Notice

The `load_repertoire`, `write_repertoire`, and `validate_repertoire` functions have been deprecated for the new generic `load_airr_data`, `write_airr_data`, and `validate_airr_data` functions. These new functions are backwards compatible with the Repertoire metadata format but also support the new AIRR objects such as GermlineSet, RepertoireGroup, GenotypeSet, Cell and Clone. This new format is defined by the DataFile Schema, which describes a standard set of objects included in a file containing AIRR Data Model presentations. Currently, the AIRR DataFile does not completely support Rearrangement, so users should continue using AIRR TSV files and its specific functions.

Also, the `repertoire_template` function has been deprecated for the `Schema.template` method, which can now be called on any AIRR Schema to create a blank object.

Reading AIRR Data Files

The `airr` package contains functions to read and write AIRR Data Model 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 AIRR objects are loaded into memory at once and no streaming interface is provided:

```
import airr

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

Why are the AIRR objects, such as Repertoire, GermlineSet, and etc., in a list versus in a dictionary keyed by their identifier (e.g., `repertoire_id`)? There are two primary reasons for this. First, the identifier might not have been assigned yet. Some systems might allow MiAIRR metadata to be entered but the identifier is assigned to that data later by another process. Without the identifier, the data could not be stored in a dictionary. Secondly, the list allows the data to have a default ordering. If you know that the data has a unique identifier then you can quickly create a dictionary object using a comprehension. For example, with repertoires:

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

another example with germline sets:

```
germline_dict = { obj['germline_set_id'] : obj for obj in data['GermlineSet'] }
```

Writing AIRR Data Files

Writing an AIRR Data File is also a light wrapper over standard YAML or JSON parsers. Multiple AIRR objects, such as Repertoire, GermlineSet, and etc., can be written together into the same file. In this example, we use the `airr` library `template` method to create some blank Repertoire objects, and write them to a file. As with the read 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
data = { 'Repertoire': [] }
for i in range(5):
    data['Repertoire'].append(airr.schema.RepertoireSchema.template())

# Write the AIRR Data
airr.write_airr('output.airr.json', data)
```

Reading AIRR Rearrangement TSV files

The `airr` package contains functions to read and write AIRR Rearrangement TSV 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 Rearrangement TSV 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')
```

By default, `create_rearrangement` will only write the `required` fields in the output file. Additional fields can be included in the output file by providing the `fields` parameter with an array of additional field names:

```
# Specify additional fields in the output
fields = ['new_calc', 'another_field']
writer = airr.create_rearrangement('output.tsv', fields=fields)
```

A common operation is to read an AIRR rearrangement file, and then write an AIRR rearrangement file with additional fields in it while keeping all of the existing fields from the original file. The `derive_rearrangement` function provides this capability:

```
import airr

# Read rearrangement data and write new file with additional fields
reader = airr.read_rearrangement('input.tsv')
fields = ['new_calc']
writer = airr.derive_rearrangement('output.tsv', 'input.tsv', fields=fields)
for row in reader:
    row['new_calc'] = 'a value'
    writer.write(row)
```

Validating AIRR data files

The `airr` package can validate AIRR Data Model JSON/YAML files and Rearrangement TSV files to ensure 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 TSV file
airr-tools validate rearrangement -a input.tsv
```

(continues on next page)

(continued from previous page)

```
# Validate an AIRR DataFile
airr-tools validate airr -a input.airr.json
```

Combining Repertoire metadata and Rearrangement files

The `airr` package does not currently keep track of which AIRR Data Model files are associated with which Rearrangement TSV files, though there is ongoing work to define a standardized manifest, so users will need to handle those associations themselves. However, in the data, AIRR identifier fields, such as `repertoire_id`, form the link between objects in the AIRR Data Model. 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 AIRR data containing repertoires
data = airr.read_airr('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
```

AIRR R Reference Library

`airr` is an R package for working with data formatted according to the AIRR Standards 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 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"`.

Reading Rearrangements

```
# Imports
library(airr)
library(tibble)

# Read Rearrangement example file
f1 <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")
rearrangement <- read_rearrangement(f1)
glimpse(rearrangement)
```

```
## Rows: 101
## Columns: 33
## $ sequence_id        <chr> "SRR765688.7787", "SRR765688.35420", "SRR765688.36681",
↪"SRR765688.33811", "SRR765688.44149", "SRR765688.15636", "SRR765688.20304", "SRR765688.
↪13860", "SR...
## $ sequence           <chr>
↪"NNNNNNNNNNNNNNNNNNNNGCTGACCTGCACCTTCTCTGGATTCTCACTCAGTACTAGTGCAGTGGGTGTACACTGGATCCGTCAGCCCCCAGGAAAGG
## $ rev_comp           <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,␣
↪FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,␣
↪FALSE, FALSE, ...
## $ productive         <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, TRUE,␣
↪FALSE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE,␣
↪TRUE, TRUE,...
## $ vj_in_frame        <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,␣
↪FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, TRUE,␣
↪TRUE, TRUE, TR...
## $ stop_codon         <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE,␣
```

(continues on next page)

(continued from previous page)

```
↪FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,
↪ FALSE, FALSE...
## $ v_call             <chr> "IGHV2-5*02", "IGHV5-51*01", "IGHV7-4-1*02", "IGHV7-4-1*02
↪", "IGHV7-4-1*02", "IGHV2-5*02", "IGHV7-4-1*02", "IGHV6-1*01,IGHV6-1*02", "IGHV7-4-1*02
↪", "IGHV4...
## $ d_call             <chr> "IGHD5-24*01", "IGHD3-16*02,IGHD3-3*01,IGHD3-3*02", "IGHD3-
↪22*01", "IGHD3-9*01", "IGHD1-26*01", "IGHD2-21*02", "IGHD1-26*01,IGHD2-21*02,IGHD3/
↪OR15-3a*01",...
## $ j_call             <chr> "IGHJ4*02", "IGHJ6*02,IGHJ6*04", "IGHJ4*02", "IGHJ6*02",
↪"IGHJ6*01", "IGHJ4*02", "IGHJ5*02", "IGHJ4*02", "IGHJ4*02", "IGHJ4*02", "IGHJ5*02",
↪"IGHJ6*02", "...
## $ c_call             <chr> "IGHG", "IGHG", "IGHG", "IGHG", "IGHG", "IGHA", "IGHA",
↪"IGHG", "IGHG", "IGHA", "IGHA", "IGHG", "IGHA", "IGHA", "IGHG", "IGHA", "IGHG", "IGHA",
↪ "IGHG", "I...
## $ sequence_alignment <chr> "..........................................................
↪.GCTGACCTGCACCTTCTCTGGATTCTCACTCAGT......
↪ACTAGTGCAGTGGGTGTACACTGGATCCGTCAGCCCCCAGGAAAGGCCCTGGAG...
## $ germline_alignment <chr> "CAGATCACCTTGAAGGAGTCTGGTCCT...
↪ACGCTGGTGAAACCCACACAGACCCTCACGCTGACCTGCACCTTCTCTGGGTTCTCACTCAGC......
↪ACTAGTGGAGTGGGTGTGGGCTGGATCCGTCAGCCCCCAGGAAAGGCCCTGGAG...
## $ junction           <chr> "TGTGCACACAGTGCGGGATGGCTGCCTGATTACTGG",
↪"TGTGCGAGGCATGGATTATACGGTTGTGATCATACCGGCTGTTATACAAGCTTCTACTACTACGGGATGGACGTCTGG",
↪"TGTGCGAGAGAAGAACGTCGAAGTAGTGGTT...
## $ junction_aa        <chr> "CAHSAGWLPDYW", "CARHGLYGCDHTGCYTSFYYYGMDVW",
↪"CAREERRSSGYFDHW", "CAREGYYFDTTGSPRSHGLDVW", "CARDSGGMDVW", "CVLSRRLGDSGVQKYYFDYW",
↪"CAREGLWDGRVVTDLW", "CAR...
## $ v_cigar            <chr>
↪"20S56N21=1X11=1X7=1X9=3X62=6D2=1X1=2X2=2X50=1X7=1X4=1X22=1X30=",
↪"20S40N15=1X15=1X11=1X2=1X1=1X1=2X3=1X7=1X41=2X2=1X10=1X3=1X1=1X5=2X5=1X4=1X9=1X19=1X24=...
## $ d_cigar            <chr> "274S5N7=", "305S29N7=", "293S13N12=", "290S9N8=",
↪"283S4N7=", "273S12N8=", "289S6N6=", "267S9N9=", "281S7N5=", "278S7N5=1X7=", "277S8N7=
↪", "297S9N7=", "2...
## $ j_cigar            <chr> "288S11N32=1X4=", "318S7N12=1X15=", "305S5N6=1X14=1X21=",
↪"321S15N5=1X23=1X17=", "290S17N19=", "296S26=1X21=", "311S11N4=1X33=",
↪"280S2N17=1X6=1X21=", "29...
## $ v_sequence_start   <int> 21, 21, 21, 21, 21, 21, 21, 20, 22, 21, 21, 20, 21, 21, 21,
↪ 21, 19, 21, 21, 21, 20, 21, 21, 21, 21, 21, 20, 23, 19, 21, 20, 21, 21, 20, 20, 21,␣
↪20, 22, 21...
## $ v_sequence_end     <int> 269, 276, 283, 283, 283, 264, 283, 259, 281, 266, 264, 294,
↪ 258, 283, 273, 279, 274, 259, 278, 280, 262, 271, 281, 262, 264, 283, 259, 279, 278,␣
↪280, 261,...
## $ v_germline_start   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪ 1, 1, 1...
## $ v_germline_end     <int> 320, 320, 320, 320, 320, 320, 320, 318, 318, 320, 320, 320,
↪ 319, 319, 317, 316, 316, 320, 318, 320, 320, 315, 318, 320, 321, 320, 318, 316, 320,␣
↪317, 319,...
## $ d_sequence_start   <int> 275, 306, 294, 291, 284, 274, 290, 268, 282, 279, 278, 298,
↪ 266, 292, 301, 285, 276, NA, 291, 284, 273, 279, 289, 277, 272, 299, 270, 292, 282,␣
↪295, 267, ...
## $ d_sequence_end     <int> 281, 312, 305, 298, 290, 281, 295, 276, 286, 291, 284, 304,
↪ 273, 296, 307, 289, 282, NA, 297, 295, 293, 290, 294, 283, 287, 307, 280, 302, 290,␣
```

(continues on next page)

(continued from previous page)

```
↪301, 280, ...
## $ d_germline_start   <int> 6, 30, 14, 10, 5, 13, 7, 10, 8, 8, 9, 10, 10, 7, 22, 14, 4,
↪ NA, 24, 4, 2, 5, 8, 9, 6, 3, 1, 3, 7, 7, 4, 6, 11, 14, 12, 8, 2, 11, 8, 10, 5, 24, 4,␣
↪17, 5, 5...
## $ d_germline_end     <int> 12, 36, 25, 17, 11, 20, 12, 18, 12, 20, 15, 16, 17, 11, 28,
↪ 18, 10, NA, 30, 15, 22, 16, 13, 15, 21, 11, 11, 13, 15, 13, 17, 11, 17, 18, 19, 13, 8,
↪ 16, 20,...
## $ j_sequence_start   <int> 289, 319, 306, 322, 291, 297, 312, 281, 300, 301, 289, 319,
↪ 276, 300, 317, 299, 296, 271, 336, 321, 303, 304, 300, 297, 293, 322, 289, 311, 315,␣
↪320, 283,...
## $ j_sequence_end     <int> 325, 346, 348, 368, 309, 344, 349, 326, 339, 347, 335, 361,
↪ 326, 334, 350, 333, 346, 320, 370, 338, 332, 338, 339, 333, 340, 368, 332, 345, 342,␣
↪362, 327,...
## $ j_germline_start   <int> 12, 8, 6, 16, 18, 1, 12, 3, 9, 2, 5, 20, 9, 14, 15, 14, 2,␣
↪13, 28, 18, 6, 14, 9, 15, 1, 16, 5, 14, 8, 5, 4, 12, 9, 1, 20, 15, 6, 6, 5, 6, 5, 9, 1,
↪ 6, 5, 5...
## $ j_germline_end     <int> 48, 35, 48, 62, 36, 48, 49, 48, 48, 48, 51, 62, 59, 48, 48,
↪ 48, 52, 62, 62, 35, 35, 48, 48, 51, 48, 62, 48, 48, 35, 47, 48, 46, 48, 44, 62, 48,␣
↪48, 51, 50...
## $ junction_length    <int> 36, 78, 45, 66, 33, 60, 48, 45, 36, 61, 51, 48, 51, 30, 54,
↪ 30, 48, 42, 71, 66, 78, 42, 36, 51, 57, 66, 51, 42, 72, 60, 45, 45, 45, 42, 36, 36,␣
↪57, 48, 51...
## $ np1_length         <int> 5, 29, 10, 7, 0, 9, 6, 8, 0, 12, 13, 3, 7, 8, 27, 5, 1, 11,
↪ 12, 3, 10, 7, 7, 14, 7, 15, 10, 12, 3, 14, 5, 4, 4, 6, 1, 7, 1, 5, 4, 5, 26, 5, 0, 15,
↪ 26, 26,...
## $ np2_length         <int> 7, 6, 0, 23, 0, 15, 16, 4, 13, 9, 4, 14, 2, 3, 9, 9, 13,␣
↪NA, 38, 25, 9, 13, 5, 13, 5, 14, 8, 8, 24, 18, 2, 22, 15, 3, 3, 11, 29, 11, 9, 5, 1, 5,
↪ 8, 0, 1, ...
## $ duplicate_count    <int> 3, 3, 13, 3, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 3, 3, 2,␣
↪2, 3, 2, 2, 2, 2, 2, 2, 5, 2, 2, 3, 2, 4, 2, 3, 4, 8, 2, 2, 2, 2, 3, 2, 4, 3, 4, 2, 5,␣
↪2, 2, 7, ...
```

Reading AIRR Data Models

AIRR Data Model records, such as Repertoire and GermlineSet, can be read from either a YAML or JSON formatted file into a nested list.

```
# Read Repertoire example file
f2 <- system.file("extdata", "repertoire-example.yaml", package="airr")
repertoire <- read_airr(f2)
glimpse(repertoire)
```

```
## List of 1
##  $ Repertoire:List of 3
##   ..$ :List of 5
##   .. ..$ repertoire_id  : chr "1841923116114776551-242ac11c-0001-012"
##   .. ..$ study          :List of 9
##   .. ..$ subject        :List of 14
##   .. ..$ sample         :List of 1
##   .. ..$ data_processing:List of 1
##   ..$ :List of 5
```

(continues on next page)

(continued from previous page)

```
##   .. ..$ repertoire_id  : chr "1602908186092376551-242ac11c-0001-012"
##   .. ..$ study          :List of 9
##   .. ..$ subject        :List of 14
##   .. ..$ sample         :List of 1
##   .. ..$ data_processing:List of 1
##   ..$ :List of 5
##   .. ..$ repertoire_id  : chr "2366080924918616551-242ac11c-0001-012"
##   .. ..$ study          :List of 9
##   .. ..$ subject        :List of 14
##   .. ..$ sample         :List of 1
##   .. ..$ data_processing:List of 1
```

```
# Read GermlineSet example file
f3 <- system.file("extdata", "germline-example.json", package="airr")
germline <- read_airr(f3)
glimpse(germline)
```

```
## List of 2
##  $ GermlineSet:List of 1
##   ..$ :List of 14
##   .. ..$ germline_set_id      : chr "OGRDB:G00007"
##   .. ..$ acknowledgements     :List of 1
##   .. ..$ release_version      : int 1
##   .. ..$ release_description  : chr ""
##   .. ..$ release_date         : chr "2021-11-24"
##   .. ..$ germline_set_name    : chr "CAST IGH"
##   .. ..$ germline_set_ref     : chr "OGRDB:G00007.1"
##   .. ..$ pub_ids              : chr ""
##   .. ..$ species              :List of 2
##   .. ..$ species_subgroup     : chr "CAST_EiJ"
##   .. ..$ species_subgroup_type: chr "strain"
##   .. ..$ locus                : chr "IGH"
##   .. ..$ allele_descriptions  :List of 2
##   .. ..$ curation             : NULL
##  $ GenotypeSet:List of 1
##   ..$ :List of 2
##   .. ..$ receptor_genotype_set_id: chr "1"
##   .. ..$ genotype_class_list     :List of 1
```

Writing AIRR formatted files

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

Writing Rearrangements

```
x1 <- file.path(tempdir(), "airr_out.tsv")
write_rearrangement(rearrangement, x1)
```

Writing AIRR Data Models

AIRR Data Model records can be written to either YAML or JSON using the `write_airr` function.

```
x2 <- file.path(tempdir(), "airr_repertoire_out.yaml")
write_airr(repertoire, x2, format="yaml")

x3 <- file.path(tempdir(), "airr_germline_out.json")
write_airr(germline, x3, format="json")
```

Validating AIRR data structures

The `airr` package contains the function `validate_rearrangement` to validate tabular (`data.frame`) Rearrangement records and AIRR Data Model objects, respectively.

```
# Validate Rearrangement data.frame
validate_rearrangement(rearrangement)
```

```
## [1] TRUE
```

```
# Validate an AIRR Data Model
validate_airr(repertoire)
```

```
## [1] TRUE
```

```
# Validate AIRR Data Model records individual
validate_airr(germline, each=TRUE)
```

```
## GenotypeSet GermlineSet
##        TRUE        TRUE
```

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_tabular

Read AIRR tabular data

Description

`read_tabular` reads a tab-delimited (TSV) file containing tabular AIRR records.

Usage

```
read_tabular(file, schema, base = c("1", "0"), aux_types = NULL, ...)
```

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

Arguments

file
: input file path.

schema
: `Schema` object defining the output format.

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).

aux_types
: named vector or list giving the type for fields that are not defined in `schema`. The field name is the name, the value the type, denoted by one of `"c"` (character), `"l"` (logical), `"i"` (integer), `"d"` (double), or `"n"` (numeric).

...
: 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.

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_tabular for writing AIRR data.

read_airr

Read an AIRR Data Model file in YAML or JSON format

Description

`read_airr` loads a YAML or JSON file containing AIRR Data Model records.

Usage

```
read_airr(
file,
format = c("auto", "yaml", "json"),
validate = TRUE,
model = TRUE
)
```

Arguments

file
: path to the input file.

format
: format of the input file. Must be one of `"auto"`, `"yaml"`, or `"json"`. If `"auto"` (default), the format will be detected from the `file` extension.

validate
: run schema validation if `TRUE`.

model
: if `TRUE` validate only AIRR DataFile defined objects. If `FALSE` attempt validation of all objects in `data`. Ignored if `validate=FALSE`

Value

A named nested `list` contained in the AIRR Data Model with the top-level names reflecting the individual AIRR objects.

Examples

```
# Get path to the Reportoire and GermlineSet example files
f1 <- system.file("extdata", "repertoire-example.yaml", package="airr")
f2 <- system.file("extdata", "germline-example.json", package="airr")

# Load data files
repertoire <- read_airr(f1)
germline <- read_airr(f2)
```

See also

See Schema for the AIRR schema definition objects. See write_airr for writing AIRR Data Model records in YAML or JSON format.

write_tabular

Write an AIRR tabular data

Description

`write_tabular` writes a TSV containing AIRR tabular records.

Usage

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

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

Arguments

data
: `data.frame` of Rearrangement data.

file
: output file name.

schema
: `Schema` object defining the output format.

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.

...
: additional arguments to pass to write_delim.

Details

`write_rearrangement` writes a `data.frame` containing AIRR Rearrangement 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)
```

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

See also

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

write_airr

Write AIRR Data Model records to YAML or JSON files

Description

`write_airr` writes a YAML or JSON file containing AIRR Data Model records.

Usage

```
write_airr(
data,
file,
format = c("auto", "yaml", "json"),
validate = TRUE,
model = TRUE
)
```

Arguments

data
: `list` containing AIRR Model Records.

file
: output file name.

format
: format of the output file. Must be one of `"auto"`, `"yaml"`, or `"json"`. If `"auto"` (default), the format will be detected from the `file` extension.

validate
: run schema validation prior to write if `TRUE`.

model
: if `TRUE` validate and write only AIRR DataFile defined objects. If `FALSE` attempt validation and write of all objects in `data`.

Examples

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

# Load data file
repertoire <- read_airr(file)

# Write a Rearrangement data file
outfile <- file.path(tempdir(), "output.yaml")
write_airr(repertoire, outfile)
```

See also

See Schema for the AIRR schema definition objects. See read_airr for reading to AIRR Data Model files.

validate_tabular

Validate tabular AIRR data

Description

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

Usage

```
validate_tabular(data, schema)
```

```
validate_rearrangement(data)
```

Arguments

data
: `data.frame` of tabular data to validate.

schema
: `Schema` object defining the data standard of the table.

Value

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

Details

`validate_rearrangement` validates the standards compliance of AIRR Rearrangement data stored in a `data.frame`

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_rearrangement(df)
```

```
[1] TRUE
```

validate_airr

Validate an AIRR Data Model nested list representation

Description

`validate_airr` validates the fields in a named nested list representation of the AIRR Data Model. Typically, generating by reading of JSON or YAML formatted AIRR files.

Usage

```
validate_airr(data, model = TRUE, each = FALSE)
```

Arguments

data
: `list` containing records of an AIRR Data Model objected imported from a YAML or JSON representation.

model
: if `TRUE` validate only AIRR DataFile defined objects. If `FALSE` attempt validation of all objects in `data`.

each
: if `TRUE` return a logical vector with results for each object in `data` instead of a single `TRUE` or `FALSE` value.

Value

Returns `TRUE` if the input `data` is compliant with AIRR standards and `FALSE` if not. If `each=TRUE` is set, then a vector with results for each each object in `data` is returned instead.

Examples

```
# Get path to the rearrangement-example file
f1 <- system.file("extdata", "repertoire-example.yaml", package="airr")
f2 <- system.file("extdata", "germline-example.json", package="airr")

# Load data file
repertoire <- read_airr(f1)
germline <- read_airr(f2)

# Validate a single record
validate_airr(repertoire)
```

```
[1] TRUE
```

```
# Return validation for individual objects
validate_airr(germline, each=TRUE)
```

```
GenotypeSet GermlineSet
       TRUE        TRUE
```

See also

See Schema for the AIRR schema definitions. See read_airr for loading AIRR Data Models from a file. See write_airr for writing AIRR Data Models to a file.

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"`
- `"Repertoire"`
- `"Study"`
- `"Subject"`
- `"Diagnosis"`
- `"Sample"`
- `"SampleProcessing"`
- `"DataProcessing"`
- `"GermlineSet"`
- `"GenotypeSet"`

Examples

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

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

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)
```

```
InfoSchema
```

```
DataFileSchema
```

```
RearrangementSchema
```

```
RepertoireSchema
```

```
GermlineSetSchema
```

```
GenotypeSetSchema
```

```
AIRRSchema
```

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.

An object of class `Schema` of length 1.

An object of class `Schema` of length 1.

An object of class `Schema` of length 1.

An object of class `Schema` of length 1.

An object of class `list` of length 25.

Details

The following predefined Schema objects are defined:

`InfoSchema`: AIRR Info `Schema`.

`DataFileSchema`: AIRR DataFile `Schema`.

`RearrangementSchema`: AIRR Rearrangement `Schema`.

`RepertoireSchema`: AIRR Repertoire `Schema`.

`GermlineSetSchema`: AIRR GermlineSet `Schema`.

`GenotypeSetSchema`: AIRR GenotypeSet `Schema`.

`AIRRSchema`: named list containing all non-experimental AIRR `Schema` objects.

Slots

`definition`
: name of the schema definition.

`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.

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.
- `extdata/repertoire-example.yaml`: Repertoire YAML file.
- `extdata/germline-example.json`: GermlineSet and GenotypeSet JSON file.

Examples

```
# Load Rearrangement example
file <- system.file("extdata", "rearrangement-example.tsv.gz", package="airr")
rearrangement <- read_rearrangement(file)
```

```
# Load Repertoire example
file <- system.file("extdata", "repertoire-example.yaml", package="airr")
repertoire <- read_airr(file)

# Load GermlineSet and GenotypeSet examples
file <- system.file("extdata", "germline-example.json", package="airr")
germline <- read_airr(file)
```

R Library Release Notes

Version 2.0.0: June 5, 2026

Deprecation and Removal:

- Completely removed `read_alignment`, `write_alignment`, and `AlignmentSchema`. Removed "Alignment" from the valid definitions list and `AIRRSchema`.

Tabular Data and Parsing:

- Updated `read_tabular` to initially treat logical columns as character data to prevent silent `NA` conversions during parsing, then explicitly casting valid logical character values.
- Updated `validate_tabular` to support character strings representing logical values (e.g., "TRUE", "T", "False", "F") when validating logical fields.

Validation:

- Enhanced `validate_entry` to support comprehensive validation of arrays without reference schemas, including item type checking (string, integer, number, boolean) and enum constraints.
- Added rigorous type and enum validation for individual fields within `validate_entry`, providing explicit warnings for unrecognized types.
- Added an ontology validation check in `validate_entry` to ensure that an ontology field is structured as a list before performing recursive validation.

Data Model and Schema:

- Updated `AIRRSchema` to replace the `Acknowledgement` schema with the new `Contributor` schema.
- Added a safety check in `extract_field_content` to handle fields with null properties safely.
-

Version 1.6.1: February 12, 2026

- Sync version with release of python airr package.

Version 1.6.0: July 7, 2025

- Updated schema set.

Version 1.5.0: August 29, 2023

- Updated schema set and examples to v1.5.

Version 1.4.1: August 27, 2022

Significant internal refactoring to improve schema generalizability, harmonize behavior between the python and R libraries, and prepare for AIRR Standards v2.0.

Rearrangement:

- Added the `aux_types` argument to `read_tabular`, `read_rearrangement`, and `read_alignment` to allow explicit declaration of the type for fields that are not defined in the schema.
- Renamed `read_airr`, `write_airr`, and `validate_airr` to `read_tabular`, `validate_tabular`, and `validate_tabular`, respectively.

Data Model and Schema:

- Defined new `read_airr`, `write_airr`, and `validate_airr` functions that support AIRR Data Model files that store arrays of objects in JSON or YAML.
- Added support for the AIRR Model Data File and associated schema (DataFile, Info). The Data File data format holds AIRR object of multiple types and is backwards compatible with Repertoire metadata.
- Added support for the new germline and genotyping schema (GermlineSet, GenotypeSet) and associated schema.

Version 1.3.0: May 26, 2020

- Updated schema set to v1.3.
- Added `info` slot to `Schema` object containing general schema information.

Version 1.2.0: August 17, 2018

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

Depends: FALSE
Imports: jsonlite, methods, readr, stats, stringi, tools, yaml
Suggests: knitr, rmarkdown, tibble, testthat

Authors

Jason Vander Heiden (aut, cre)
Susanna Marquez (aut)
Scott Christley (aut)
Katharina Imkeller (aut)
Ulrik Stervbo (aut)
Gisela Gabernet (aut)
AIRR Community (cph)

License

CC BY 4.0

AIRR JavaScript Reference Library

The `airr-js` reference library provides basic functions and classes for interacting with AIRR Community Data Representation Standards, including tools for read, write and validation. The library can be used in the browser or nodejs.

JavaScript Library Release Notes

Version 2.0.0: June 5, 2026

Initial release.

Installation

Install in the usual manner from npm:

```
> npm install airr-js
```

Or from the downloaded source code directory:

```
> npm install file:lang/js
```

Quick Start

The `airr-js` package supports use in the browser or in nodejs. In the browser, the file system is not available, so the read/write functions are not implemented but template objects can be created, objects can be validated, and the OpenAPI V3 specification can be accessed. For nodejs, the full functionality is available including the read/write functions.

For nodejs, need to await the loading of the schema before using any functions:

```
var airr = require('airr-js');

// await schema to be loaded and dereferenced
var spec = await airr.load_schema();
```

For the browser, the schema also needs to be loaded but the package cannot do it itself, instead you must provide the Open API V3 specification file as part of your packaging of the website. When using webpack, a resolve alias in `webpack.config.js` can be used to point to the dereferenced yaml file:

```
resolve: {
    alias: {
        'airr-schema': path.resolve(__dirname,'node_modules') + '/airr-js/airr-schema-
↪openapi3-deref.yaml'
    }
}
```

The `package.json` utilizes the `browser` setting supported by website packaging tools like webpack to provide an alternative entry point, and browser code can import like so:

```
import { airr } from 'airr-js';
```

The read and write functions for AIRR Rearrangement TSV files support gzip compressed data. File names that end with `.gz` extension will automatically be uncompressed when reading or automatically compressed when writing.

Create Blank Template Schema Objects (browser, nodejs)

Import the `airr-js` package correctly depending upon browser or nodejs usage as described above, and then blank template objects can be created:

```
// Get the schema definition for an AIRR Object
var repertoire_schema = new airr.SchemaDefinition('Repertoire');
// Create a template object
var blank_repertoire = repertoire_schema.template();
```

Validate Objects (browser, nodejs)

Import the `airr-js` package correctly depending upon browser or nodejs usage as described above, and then an object can be validated to its schema:

```
// Get the schema definition for an AIRR Object
var repertoire_schema = new airr.SchemaDefinition('Repertoire');
// Validate a repertoire object
var is_valid = repertoire_schema.validate_object(obj);
```

Reading AIRR Data Files (nodejs)

The `airr-js` package contains functions to read and write AIRR Data Model 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 AIRR objects are loaded into memory at once and no streaming interface is provided:

```
var airr = require('airr-js');
await airr.load_schema();

// read AIRR DataFile
var data = await airr.read_airr('input.airr.json');
```

Writing AIRR Data Files (nodejs)

Writing an AIRR Data File is also a light wrapper over standard YAML or JSON parsers. Multiple AIRR objects, such as Repertoire, GermlineSet, and etc., can be written together into the same file. In this example, we create some blank Repertoire objects, and write them to a file. As with the read function, the complete list of repertoires are written at once, there is no streaming interface:

```
var airr = require('airr-js');
await airr.load_schema();

// Create some blank repertoire objects in a list
var repertoire_schema = new airr.SchemaDefinition('Repertoire');
var data = { 'Repertoire': [] };
for (let i = 0; i < 5; ++i)
    data['Repertoire'].push(repertoire_schema.template());

// Write the AIRR Data
await airr.write_airr('output.airr.json', data);
```

Reading AIRR Rearrangement TSV files (nodejs)

The `airr-js` package contains functions to read AIRR Rearrangement TSV files as either a stream or the complete file. The streaming interface requires two callback functions to be provided; one for the header and another for each row as it is read. The callback functions can be synchronous or they can return a Promise:

```
var airr = require('airr-js');
await airr.load_schema();

// read file completely
var data = await airr.load_rearrangement('input.airr.tsv');
```

(continues on next page)

(continued from previous page)

```
// for streaming, need two callback functions
var header_callback = function(headers) { console.log('got headers:', headers); }
var row_callback = function(row) { console.log('got row:', row); }
// read the file
await airr.read_rearrangement('input.airr.tsv', header_callback, row_callback);
```

Writing AIRR Rearrangement TSV files (nodejs)

The `airr-js` package contains functions to write AIRR Rearrangement TSV files as either a stream or the complete file. The streaming interface requires a callback function which provides the data for each row or returns `null` to indicate no more data. The callback function can be synchronous or it can return a Promise:

```
var airr = require('airr-js');
await airr.load_schema();

// read some data
var data = await airr.load_rearrangement('input.airr.tsv');

// write file completely
var data = await airr.load_rearrangement(data, 'output.airr.tsv');

// for streaming, need a callback function to provide the row data
var idx = 0;
var row_callback = function(fields) {
    if (idx >= data.length) return null;
    else return data[idx++];
};
// write the file
await airr.create_rearrangement('output.airr.tsv', row_callback)

// callback function which returns a promise
var row_callback = function(fields) {
    return new Promise(function(resolve, reject) {
        // acquire some data asynchronously, e.g. from a database
        row = await read_from_database();
        return resolve(row);
    });
};
```

2.6 AIRR Data Commons API V1

Note: This section is about the API, if you are looking for information on the AIRR Data Commons, please see *here*.

2.6.1 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.

2.6.2 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.

Repository implementation principles

Implementers of the ADC API should follow the following high level principles. Users of the ADC API can expect the following principles to be followed.

- All API endpoints should return JSON encodings as an API response.
- For some API endpoints it is possible to request TSV files, and those endpoints that support TSV files are documented in the *API Endpoints* section.
- Endpoints that are not documented as supporting TSV can reject TSV requests.
- If an API endpoint returns a field, then the content of that field in the JSON and TSV response must be equivalent.
- For those fields that contain Amino Acid or Nucleotide strings, the case for the characters (upper or lower case) is not stated in the specification. Repository implementations should expect upper or lower case queries for these fields. Repositories may want to enforce internal characteristics for these fields (e.g. AA are always upper case, nt are always lower case) to facilitate efficient storage and searching. Because case is not stated, repositories can return amino acid and nucleotide sequences using the case utilized internally.
- Relevant HTTP error codes should be returned on error conditions. HTTP 408 (timeout) should be used if the API does not complete an operation because of an internal time limit, and HTTP 413 (Content too large) should be returned when either max_size or max_query_size are exceeded.
- Extensions beyond the standard API, e.g., support for the Async API, should be specified with the *extensions* property in the */info* endpoint.

Repository operation principles

Research groups that are running repositories as part of the AIRR Data Commons should, to the best of their ability, ensure that their repository uptime is maintained and that repository queries on fields that have the adc_query_support attribute set are completed in a timely manner.

In order to maximize scientific reproducibility and data provenance, it is recommended that data stewards/data curators avoid releasing partially loaded data into the AIRR Data Commons. When loading a study it is recommended that all data from a specific AIRR Schema object (e.g. Rearrangement, Clone, Cell) be loaded and then made accessible in the ADC as a single package, rather than having the repository accessible in the ADC while the data is being loaded. Piecemeal data loading of data for a specific schema object (e.g. Rearrangement) for a study in a production repository will result in queries returning different results as searches are made over time. This can lead to consumers of the data receiving confusing results, makes for complicated data provenance, and hampers scientific reproducibility.

Authentication

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

Extensions

Implementation of the ADC API is sufficient for most repositories. However, repositories may also implement extension APIs that provide additional capability and functionality.

- Asynchronous API.
- Statistics API (for future consideration).

2.6.3 Table of Content

API Endpoints

The ADC API is versioned with the version number (`v1`) as part of the base path for all endpoints. The ADC API provides a query endpoint for each of the primary, high-level objects in the AIRR Standard to retrieve AIRR-seq related data about that type of object. 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 other objects such as `Rearrangement`, `Clone`, `Cell`, `CellExpression`, `CellReactivity`, and `Receptor`, are provide respectively by the `rearrangement`, `clone`, `cell`, `expression`, `reactivity`, and `receptor` endpoints.

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 other object endpoints (e.g. `rearrangement`) to download the data of that type 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.

Each ADC API endpoint provides 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*.
/v1/clone/{clone_id}	Retrieve a Clone given its clone_id	GET	Upon success, returns the Clone information in JSON format according to the *Clone schema*.
/v1/clone	Query clones	POST	Upon success, returns a list of Clones in JSON format according to the *Clone schema*.
/v1/cell/{cell_id}	Retrieve a Cell given its cell_id	GET	Upon success, returns the Cell information in JSON format according to the *Cell schema*.
/v1/cell	Query cells	POST	Upon success, returns a list of Cells in JSON format according to the *Cell schema*.
/v1/expression/{expression_id}	Retrieve a Expression Property given its expression_id	GET	Upon success, returns the Expression information in JSON format according to the CellExpression schema.
/v1/expression	Query Cell Expression properties	POST	Upon success, returns a list of Expression Properties in JSON format according to the CellExperssion schema.
/v1/receptor/{receptor_id}	Retrieve a Receptor given its receptor_id	GET	Upon success, returns the Receptor information in JSON format according to the *Receptor schema*.
/v1/receptor	Query Receptor properties	POST	Upon success, returns a list of Receptors in JSON format according to the *Receptor schema*.
/v1/reactivity/{cell_reactivity_	Retrieve a CellReactivity given its cell_reactivity_id	GET	Upon success, returns the CellReactivity information in JSON format according to the CellReactivity schema.
/v1/reactivity	Query CellReactivity properties	POST	Upon success, returns a list of CellReactivity objects in JSON format according to the CellReactivity schema.

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/5993695857891348971-242ac118-0001-012
```

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


```
{
  "Info": {
    "title": "AIRR Data Commons API for VDJServer Community Data Portal",
    "description": "VDJServer ADC API response for repertoire query",
    "version": "1.3",
    "contact": {
      "name": "VDJServer",
      "url": "http://vdjserver.org/",
      "email": "vdjserver@utsouthwestern.edu"
    }
  },
  "Repertoire": [
    {
      "repertoire_id": "5993695857891348971-242ac118-0001-012",
      "repertoire_name": null,
      "repertoire_description": null,
      "study": {
        "study_id": "4995411523885404651-242ac118-0001-012",
        "study_title": "T cell Receptor Repertoires Acquired via Routine Pap Testing May␣
↪Help Refine Cervical Cancer and Precancer Risk Estimates",
        "study_type": {
          "id": "NCIT:C16084",
          "label": "Observational Study"
        },
        "study_description": "Cervical cancer is the fourth most common cancer and␣
↪fourth leading cause of cancer death among women worldwide. In low Human Development␣
↪Index settings, it ranks second. Screening and surveillance involve the cytology-based␣
↪Papanicolaou (Pap) test and testing for high-risk human papillomavirus (hrHPV). The␣
↪Pap test has low sensitivity to detect precursor lesions, while a single hrHPV test␣
↪cannot distinguish a persistent infection from one that the immune system will␣
↪naturally clear. Furthermore, among women who are hrHPV-positive and progress to high-
↪grade cervical lesions, testing cannot identify the ~20% who would progress to cancer␣
↪if not treated. Thus, reliable detection and treatment of cancers and precancers␣
↪requires routine screening followed by frequent surveillance among those with past␣
↪abnormal or positive results. The consequence is overtreatment, with its associated␣
↪risks and complications, in screened populations and an increased risk of cancer in␣
↪under-screened populations. Methods to improve cervical cancer risk assessment,␣
↪particularly assays to predict regression of precursor lesions or clearance of hrHPV␣
↪infection, would benefit both populations. Here we show that women who have lower risk␣
↪results on follow-up testing relative to index testing have evidence of enhanced T␣
↪cell clonal expansion in the index cervical cytology sample compared to women who␣
↪persist with higher risk results from index to follow-up. We further show that a␣
↪machine learning classifier based on the index sample T cells predicts this transition␣
↪to lower risk with 95% accuracy (19/20) by leave-one-out cross-validation. Using T␣
↪cell receptor deep sequencing and machine learning, we identified a biophysicochemical␣
↪motif in the complementarity-determining region 3 of T cell receptor  chains whose␣
↪presence predicts this transition. While these results must still be tested on an␣
↪independent cohort in a prospective study, they suggest that this approach could␣
↪improve cervical cancer screening by helping distinguish women likely to spontaneously␣
```


(continues on next page)

(continued from previous page)


```
↪regress from those at elevated risk of progression to cancer. The advancement of such␣
↪a strategy could reduce surveillance frequency and overtreatment in screened␣
↪populations and improve the delivery of screening to under-screened populations.",
       "inclusion_exclusion_criteria": "We included samples from White Hispanic women␣
↪age 18 years or older. We excluded women who were HIV+, pregnant, had an intrauterine␣
↪device, or had a sexually transmitted disease at the time of sample collection. We␣
↪obtained samples across all cytology result categories: Negative for Intraepithelial␣
↪Lesion or Malignancy (NILM, Normal), Abnormal Squamous Cells of Undetermined␣
↪Significance (ASCUS), Low-grade Squamous Intraepithelial Lesion (LSIL), and High-grade␣
↪Squamous Intraepithelial Lesion (HSIL). At Parkland Health and Hospital System (PHHS),␣
↪the primary screening strategy is cytology alone with a reflex hrHPV test for women␣
↪with an ASCUS cytology result. The test assays for positivity across 14 HPV types, and␣
↪the ASCUS result category is divided into ASCUS/HPV- (negative for all 14 types) and␣
↪ASCUS/HPV+ (positive for at least one type). An additional exclusion criterion was␣
↪applied to women with a result of Normal, ASCUS/HPV-, and ASCUS/HPV+, and that is they␣
↪were excluded if they had previously had cervical cancer or previous treatment of␣
↪cervical pre-cancerous lesions.\n\nWe applied these inclusion and exclusion criteria␣
↪in a quota sampling scheme to ensure adequate representation of women across all five␣
↪result categories. We targeted a minimum of 100 samples total with a minimum of 15␣
↪samples in each category, and then rescued all samples meeting our criteria each week␣
↪until all minimums were reached.",
       "lab_name": "Lindsay G. Cowell",
       "lab_address": "UT Southwestern Medical Center",
       "submitted_by": "Scott Christley, scott.christley@utsouthwestern.edu, UT␣
↪Southwestern Medical Center",
       "grants": "This research was supported by Simmons Comprehensive Cancer Center␣
↪Development Funds and by a charitable donation from Young Texans Against Cancer, both␣
↪to LC and JT.",
       "pub_ids": "PMID: 33868241",
       "keywords_study": [
         "contains_tcr"
       ],
       "adc_publish_date": "2021-08-05T03:50:02.295Z",
       "adc_update_date": "2021-08-05T05:43:14.260Z",
       "collected_by": null
     },
     "subject": {
       "subject_id": "5_20",
       "synthetic": false,
       "species": {
         "id": "NCBITAXON:9606",
         "label": "Homo sapiens"
       },
       "sex": "female",
       "age_min": 49.1,
       "age_max": 49.1,
       "age_unit": {
         "id": "UO:0000036",
         "label": "year"
       },
       "ethnicity": "Hispanic",
       "race": "White",
```


(continues on next page)

(continued from previous page)


```
      "diagnosis": [
        {
          "disease_diagnosis": {
            "id": null,
            "label": null
          },
          "study_group_description": null,
          "disease_length": null,
          "disease_stage": null,
          "prior_therapies": null,
          "immunogen": null,
          "intervention": null,
          "medical_history": null
        }
      ],
      "age_event": null,
      "ancestry_population": null,
      "strain_name": null,
      "linked_subjects": null,
      "link_type": null
    },
    "sample": [
      {
        "sample_id": "5_20_DNA",
        "sample_type": "cytology",
        "tissue": {
          "id": "UBERON:0004801",
          "label": "cervix epithelium"
        },
        "anatomic_site": "cervix",
        "disease_state_sample": "hsil",
        "cell_species": {
          "id": null,
          "label": null
        },
        "single_cell": false,
        "cell_storage": false,
        "template_class": "DNA",
        "template_amount": "2ug",
        "library_generation_method": "PCR",
        "library_generation_protocol": "Adaptive Biotechnologies",
        "library_generation_kit_version": "v3",
        "pcr_target": [
          {
            "pcr_target_locus": "TRB",
            "forward_pcr_primer_target_location": null,
            "reverse_pcr_primer_target_location": null
          }
        ],
        "complete_sequences": "partial",
        "physical_linkage": "none",
        "sequencing_run_id": "UTSW-Monson-P02-04",
```


(continues on next page)

(continued from previous page)


```
          "sequencing_run_date": "11/16/17",
          "sequencing_files": {
            "file_type": "fasta",
            "filename": "5-20_DNA.fasta",
            "read_direction": "forward",
            "read_length": null,
            "paired_filename": null,
            "paired_read_direction": null,
            "paired_read_length": null
          },
          "sample_processing_id": null,
          "collection_time_point_relative": null,
          "collection_time_point_reference": null,
          "biomaterial_provider": null,
          "tissue_processing": null,
          "cell_subset": {
            "id": null,
            "label": null
          },
          "cell_phenotype": null,
          "cell_number": null,
          "cells_per_reaction": null,
          "cell_quality": null,
          "cell_isolation": null,
          "cell_processing_protocol": null,
          "template_quality": null,
          "total_reads_passing_qc_filter": null,
          "sequencing_platform": null,
          "sequencing_facility": null,
          "sequencing_kit": null
        }
      ],
      "data_processing": [
        {
          "data_processing_id": "bf0617e7-b4a4-480f-99e3-b53eef9ca6d4-007",
          "primary_annotation": true,
          "software_versions": "igblast-ls5-1.14u2",
          "data_processing_files": [
            "5-20_DNA.igblast.airr.tsv.gz"
          ],
          "germline_database": "VDJServer IMGT 2019.01.23",
          "paired_reads_assembly": null,
          "quality_thresholds": null,
          "primer_match_cutoffs": null,
          "collapsing_method": null,
          "data_processing_protocols": null,
          "analysis_provenance_id": null
        }
      ]
    }
  ]
}
```

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 -H 'content-type: application/json' https://
↪vdjserver.org/airr/v1/repertoire
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
                "content": {
                    "field":"subject.species.id",
                    "value":"NCBITAXON: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/610b77f6d5812c007f79bba3
```

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

```
{
  "Info":
  {
```

(continues on next page)

(continued from previous page)

```
      "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":"610b77f6d5812c007f79bba3",
      "repertoire_id":"5993695857891348971-242ac118-0001-012",
      "data_processing_id": "bf0617e7-b4a4-480f-99e3-b53eef9ca6d4-007",

      "... 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 -H 'content-type: application/json' https://
↪vdjserver.org/airr/v1/rearrangement
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":[
                        "2603354229190496746-242ac113-0001-012",
                        "2618085967015776746-242ac113-0001-012",
                        "2633633748627296746-242ac113-0001-012",
                        "2564613624180576746-242ac113-0001-012"
                    ]
                }
            },
            {
                "op":"=",
                "content": {
```

(continues on next page)

(continued from previous page)

```
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
            }
        ]
    },
    "fields":["repertoire_id","sequence_id","v_call","productive"],
    "format":"tsv"
}
```

Here is the response in AIRR TSV format.

```
sequence_id productive      v_call  repertoire_id
5f70b421e10383007e3038ad    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e3038c2    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e3038f0    true    IGHV1-69*10     2564613624180576746-242ac113-0001-012
5f70b421e10383007e3039ec    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303a1b    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303a22    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303a23    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303a47    true    IGHV1-24*01     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303b00    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
5f70b421e10383007e303baf    true    IGHV1-69*04     2564613624180576746-242ac113-0001-012
```

Clone Endpoint

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

Unlike repertoire data, data repositories are expected to store millions or billions of clone 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 Clone

Given a `clone_id`, a single `Clone` object will be returned.

```
curl https://covid19-1.ireceptor.org/airr/v1/clone/{clone_id}
```

Where clone_id is the ID of a clone object in the repository. The response will provide the `Clone` data in JSON format.

```
{
  "Info":
  {
    "title": "airr-api-ireceptor",
    "description": "AIRR Data Commons API for iReceptor",
    "version": "3.0",
    "last_update": null,
    "contact": {
        "name": "iReceptor",
        "url": "http://www.ireceptor.org",
        "email": "support@ireceptor.org"
    }
  },
```

(continues on next page)

(continued from previous page)


```
  "Clone":
  [
    {
      "clone_id": "clonotype1",
      "repertoire_id": "PRJCA002413-Healthy_Control_1-IG",
      "data_processing_id": "PRJCA002413-Healthy_Control_1",
      "sequences": null,
      "v_call": "IGHV2-70",
      "d_call": "",
      "j_call": "IGHJ3",
      "junction": "TGCGCACGGGCTCATTGTTCGTGGGGCAGCAGCAGGTTCGGTGCTTTTGATATGTGG",
      "junction_aa": "CARAHCSWGSSRFGAFDMW",
      "junction_length": 57,
      "junction_aa_length": 19,
      "FIELDS REMOVED" : "FOR SPACE"
    }
  ]
}
```


Query against all Clones

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 clones 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 is provided in JSON format.

```
curl -d '{"filters":{"op":"=","content":{"field":"junction_aa","value":
↪"CARAHCSWGSSRFGAFDMW"}},"size":1}' -H 'content-type: application/json' http://covid19-
↪1.ireceptor.org/airr/v1/clone
```

This query searches the repository for clones that have a specific `junction_aa` field with a value of `CARAHCSWGSSRFGAFDMW` and requests only a single object in the response (`"size":1`). The response would be similar to that provided by the single clone query given above.

Cell Endpoint

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

Unlike repertoire data, data repositories are expected to store millions of cell 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 Cell

Given a `cell_id`, a single `Cell` object will be returned.

```
curl https://covid19-1.ireceptor.org/airr/v1/cell/{cell_id}
```

Where cell_id is the ID of a cell object in the repository. The response will provide the `Cell` data in JSON format.


```
{"Info":{
   "title": "airr-api-ireceptor",
   "description": "AIRR Data Commons API for iReceptor",
```


(continues on next page)

(continued from previous page)


```
  "version": "3.0",
  "last_update": null,
  "contact": {
      "name": "iReceptor",
      "url": "http://www.ireceptor.org",
      "email": "support@ireceptor.org"
  }
}, "Cell":[
{
  "cell_id": "AAACCTGCACCGATAT-1",
  "rearrangements": null,
  "receptors": null,
  "reactivity_measurements": [ "61fc6c454f24ed3af5456a54" ],
  "repertoire_id": "PRJCA002413-ERS1-CELL",
  "data_processing_id": "PRJCA002413-ERS1",
  "expression_study_method": "single-cell transcriptome",
  "expression_raw_doi": null,
  "expression_index": null,
  "virtual_pairing": false
}]}
```


Query against all Cells

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 clones 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 is provided in JSON format.

```
curl -d '{"filters":{"op":"=","content":{"field":"repertoire_id","value":"PRJCA002413-
↪ERS1-CELL"}},"size":1}' -H 'content-type: application/json' http://covid19-1.ireceptor.
↪org/airr/v1/cell
```

This query searches the repository for cells that have a specific `repertoire_id` field with a value of `PRJCA002413-ERS1-CELL` and requests only a single object in the response (`"size":1`). The response would be similar to that provided by the single cell query given above.

Expression Endpoint

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

Unlike repertoire data, data repositories are expected to store millions or billions of cell expression 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 Cell Expression Property

Given a `expression_id`, a single `Expression` object will be returned.

```
curl https://covid19-1.ireceptor.org/airr/v1/expression/{expression_id}
```

Where expression_id is the ID of an expression object in the repository. The response will provide the `CellExpression` data in JSON format.

```
{"Info":{
  "title": "airr-api-ireceptor",
  "description": "AIRR Data Commons API for iReceptor",
  "version": "3.0",
  "last_update": null,
  "contact": {
      "name": "iReceptor",
      "url": "http://www.ireceptor.org",
      "email": "support@ireceptor.org"
  }
}, "CellExpression":[
{
  "expression_id": "61fc6c454f24ed3af5456a54",
  "cell_id": "AAACCTGCAGCTTAAC-1",
  "repertoire_id": "PRJCA002413-Healthy_Control_1-CELL",
  "data_processing_id": "PRJCA002413-Healthy_Control_1",
  "property": {
      "label": "ISG15",
      "id": "ENSG:ENSG00000187608"
  },
  "value": 1
}]}
```


Query against all Cell Expression data

Supplying a `repertoire_id` or `cell_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 cell expression data with an ENSEMBL gene ID with the value `ENSG:ENSG0000017575` and requests only a single object response (`"size":1`). The resultant data is provided in JSON format and would be similar to that provided by the single expression property query given above.

```
curl -d '{"filters":{"op":"=","content":{"field":"property.id","value":
↪"ENSG:ENSG00000175756"}},"size":1}' -H 'content-type: application/json' http://covid19-
↪1.ireceptor.org/airr/v1/expression
```

CellReactivity Endpoint

The `reactivity` endpoint provides access to all fields in the CellReactivitySchema. There are two type of endpoints: One for retrieving a single reactivity object given its identifier, and another for performing a query across all cell reactivity in the data repository.

To allow data repositories to optimize their databases for performance, the ADC API does not require that all fields in the `CellReactivity` object to be queryable and only a limited set of query capabilities must be supported. The queryable fields are described in the Fields section below.

Retrieve CellReactivity data

Given a `cell_reactivity_id`, a single `CellReactivity` object will be returned.

```
curl https://covid19-1.ireceptor.org/airr/v1/reactivity/{cell_reactivity_id}
```

Where `cell_reactivity_id` is the ID of a `CellReactivity` object in the repository. The response will provide the object in JSON format.

```
{"Info": {
  "title": "airr-api-ireceptor",
  "description": "AIRR Data Commons API for iReceptor",
  "version": "4.0",
  "last_update": null,
  "contact": {
      "name": "iReceptor",
      "url": "http://www.ireceptor.org",
      "email": "support@ireceptor.org"
  }
}, "CellReactivity": [
  {
    "cell_reactivity_id": "61fc6c454f24ed3af5456a54",
    "ligand_type" : "MHC:peptide",
    "antigen_type" : "peptide",
    "antigen" : { "id" : "NCBI:YP_009725307.1", "label" : "RNA-dependent RNA polymerase␣
↪(nsp12)"},
    "antigen_source_species" : { "id" : "NCBITAXON:2697049", "label" : "Severe acute␣
↪respiratory syndrome coronavirus 2"},
    "peptide_start" : 738,
    "peptide_end" : 746,
    "peptide_sequence_aa" : "DTDFVNEFY",
    "mhc_class" : "MHC-I",
    "mhc_gene_1       id:" : "MRO:0000046, label: HLA-A",
    "mhc_allele_1" : "HLA-A*01:01",
    "reactivity_method" : "dextramer barcoding",
    "reactivity_readout" : "barcode count",
    "reactivity_value" : 23,
    "reactivity_unit" : "absolute count"
  }]
}
```


Query against all CellReactivity data

This example queries for receptor reactivity data that has a `peptide_sequence_aa` of `DTDFVNEFY` and requests only a single object of that type (`"size":1`). The resultant data is provided in JSON format and would be similar to that provided by the cell reactivity query response given above.

```
curl -d '{"filters":{"op":"=","content":{"field":"peptide_sequence_aa","value":"DTDFVNEFY
↪"}},"size":1}' -H 'content-type: application/json' http://covid19-1.ireceptor.org/airr/
↪v1/reactivity
```

Receptor Endpoint

The `receptor` endpoint provides access to all fields in the *Receptor Schema*. There are two type of endpoints: One for retrieving a single receptor given its identifier, and another for performing a query across all receptors in the data repository.

To allow data repositories to optimize their databases for performance, the ADC API does not require that all fields in the `Receptor` object to 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 Receptor

Given a `receptor_id`, a single `Receptor` object will be returned.

```
curl https://covid19-1.ireceptor.org/airr/v1/receptor/{receptor_id}
```

Where `receptor_id` is the ID of a `Receptor` object in the repository. The response will provide the object in JSON format.


```
{"Info": {
  "title": "airr-api-ireceptor",
  "description": "AIRR Data Commons API for iReceptor",
  "version": "3.0",
  "last_update": null,
  "contact": {
      "name": "iReceptor",
      "url": "http://www.ireceptor.org",
      "email": "support@ireceptor.org"
  }
}, "Receptor": [
  {
    "receptor_id": "IG-MM-BALB-123456",
    "receptor_hash": "aa1c4b77a6f4927611ab39f5267415beaa0ba07a952c233d803b07e52261f026",
    "receptor_type": "Ig",
    "receptor_variable_domain_1_aa":
↪"QVQLQQPGAELVKPGASVKLSCKASGYTFTSYWMHWVKQRPGRGLEWIGRIDPNSGGTKYNEKFKSKATLTVDKPSSTAYMQLSSLTSEDSAVYYCARYD
↪",
    "receptor_variable_domain_1_locus": "IGH",
    "receptor_variable_domain_2_aa":
↪"QAVVTQESALTTSPGETVTLTCRSSTGAVTTSNYANWVQEKPDHLFTGLIGGTNNRAPGVPARFSGSLIGDKAALTITGAQTEDEAIYFCALWYSNHWVF
↪",
    "receptor_variable_domain_2_locus": "IGL",
    "receptor_ref": [ "IEDB_RECEPTOR:29263" ],
    "reactivity_measurements": [ "61fc6c454f24ed3af5456a54" ]
  }
]}
```


Query against all Receptor data

This example queries for receptor data that has a TCR receptor type and requests only a single object response (`"size":1`). The resultant data is provided in JSON format and would be similar to that provided by the single expression property query given above.

```
curl -d '{"filters":{"op":"=","content":{"field":"receptor_type","value":"TCR"}},"size
↪":1}' -H 'content-type: application/json' http://covid19-1.ireceptor.org/airr/v1/
↪receptor
```

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/ADC fields
miairr	Y	some	N	N
airr-core	Y	Y	Y	N
airr-schema	Y	Y	Y	Y

The sets included in `include_fields` are:

- `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.

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/info
```

The response provides various information about the repository itself, the API that is implemented by the repository, the schema version of the data that is returned from repository queries, as well as repository specific details such as the maximum number of records that are allowed to be requested in a single query as well as the maximum size (in bytes) of the queries sent to the repository.

```
{
  "title": "api-js-tapis",
  "description": "AIRR Data Commons API for VDJServer Community Data Portal",
  "version": "2.0.0",
  "contact": {
    "name": "VDJServer",
    "url": "http://vdjserver.org/",
    "email": "vdjserver@utsouthwestern.edu"
  },
```

(continues on next page)

(continued from previous page)

```
  "license": {
    "name": "GNU AGPL V3"
  },
  "api": {
    "title": "AIRR Data Commons API",
    "version": "1.0.0",
    "contact": {
      "name": "AIRR Community",
      "url": "http://www.airr-community.org/",
      "email": "join@airr-community.org"
    },
    "description": "Major Version 1 of the Adaptive Immune Receptor Repertoire (AIRR)␣
↪data repository web service application programming interface (API).\n",
    "license": {
      "name": "Creative Commons Attribution 4.0 International",
      "url": "https://creativecommons.org/licenses/by/4.0/"
    }
  },
  "schema": {
    "title": "AIRR Schema",
    "description": "Schema definitions for AIRR standards objects",
    "version": "1.3",
    "contact": {
      "name": "AIRR Community",
      "url": "https://github.com/airr-community"
    },
    "license": {
      "name": "Creative Commons Attribution 4.0 International",
      "url": "https://creativecommons.org/licenses/by/4.0/"
    }
  },
  "max_size": 1000,
  "max_query_size": 2097152,
  "extensions": ["async_api"]
}
```

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 -H 'content-type: application/json' https://
↪vdjserver.org/airr/v1/repertoire
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"=",
```

(continues on next page)

(continued from previous page)

```
                "content": {
                    "field":"subject.species.id",
                    "value":"NCBITAXON:9606"
                }
            },
            {
                "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": "5993695857891348971-242ac118-0001-012"},
      {"repertoire_id": "5981154557681996267-242ac118-0001-012"},
      {"repertoire_id": "6018649617881108971-242ac118-0001-012"},
      {"repertoire_id": "5959121371158548971-242ac118-0001-012"},
      {"repertoire_id": "5939278622251028971-242ac118-0001-012"}
  ]
}
```

Request Parameters

The ADC API supports the follow query parameters. These are only applicable to the 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	Operand	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 not missing	field	n/a	field is not missing and is not null	{“op”:”is not missing”,”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”,”NO
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 operators	n/a	logical AND	{“op”:”and”,”content”:[{“op”:”!=”,”content”:{“field”:”subject.organism.id”,”value”:”960 {“op”:”>=”,”content”:{“field”:”sample.cells_per_reaction”,”value”:10000}}, {“op”:”exclude”,”content”:{“field”:”subject.strain_name”,”value”:[“SCID”,”NOD”]}}]}
or	multiple operators	n/a	logical OR	{“op”:”or”,”content”:[{“op”:”<”,”content”:{“field”:”sample.cell_number”,”value”:1000} {“op”:”is missing”,”content”:{“field”:”sample.tissue”}}, {“op”:”exclude”,”content”:{“field”:”subject.organism.id”,”value”:[“9606”,”10090”]}}]}

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.

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"]
        }
      }
    ]
  }
}
```

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. A specific

ordering of fields in the TSV format should not be assumed from one API request to another. Take care to properly merge AIRR TSV data from multiple API requests, e.g. such as with the `airr-tools merge` program.

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 -H 'content-type: application/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 for VDJServer Community Data Portal",
    "description": "VDJServer ADC API response for repertoire query",
    "version": "1.3",
    "contact": {
      "name": "VDJServer",
      "url": "http://vdjserver.org/",
      "email": "vdjserver@utsouthwestern.edu"
    }
  },
  "Facet": [
    {
      "sample.pcr_target.pcr_target_locus": "TRB",
      "count": 2786
    },
    {
      "sample.pcr_target.pcr_target_locus": "TRA",
      "count": 242
    },
    {
      "sample.pcr_target.pcr_target_locus": "IGK",
      "count": 122
    },
    {
      "sample.pcr_target.pcr_target_locus": "IGH",
      "count": 547
    },
    {
      "sample.pcr_target.pcr_target_locus": "IGL",
      "count": 121
    }
  ]
}
```

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":"and",
        "content": [
```

(continues on next page)

(continued from previous page)

```
            {
                "op":"=",
                "content": {
                    "field":"study.study_id",
                    "value":"PRJNA300878"
                }
            },
            {
                "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 -H 'content-type: application/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}
  ]
}
```

Note: ADC API facet requests differ from those in the GDC API on which the ADC API is based. In the ADC API it is allowed to request a facet count on a field that is being filtered, whereas in the GDC API filters on the facet'ed field are ignored (see Genomic Data Commons (GDC) API Facets restriction #2).

Queries on Nested Information (Arrays)

As stated above, in general API response data will be have been flattened by the query handler. However, there are several instances in which properties within the top-level entities are arrays of objects, which cannot be flattened because all the information will be expected to present in the response. Therefore, in these cases, the data that is queried and potentially returned will be nested. In addition, while the array of object is obvious from the AIRR Schema, the array component (index) does **not** appear in the hierarchical property names used by the API. Note that this does not create any collisions as the schema allows the existence of multiple properties with the same designation.

However, it results in two possible ways how an `AND` operator can behave when using such nested properties as input. These are defined as follows:

Given that two or more instances of an object class exists within an array that is a property of a higher-level entity, and given a query that contains an AND operation that uses tests on two or more properties of said object class as an input, the code handling the query will exhibit

- *local* behavior, if for the `AND` operation to evaluate to `TRUE` it requires that all tests must succeed within an instance of the object and within at least one object of the array, i.e., the code is aware of the nesting and is able to parse the hierarchy of properties from the provided string, or
- *global* behavior, if for the `AND` operation to evaluate to `TRUE` it requires all tests to succeed, but independent of the instances in which the matching properties are located, i.e., the code is agnostic to the nesting and treats all properties within the array as a single set.

While both behaviors have their use cases, ADC API handlers are expected to exhibit "local" behavior, as is easier to implement on the client-side, where it would require joining the the result sets of the queries for each of the properties individually.

An example query 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.

Given the example diagnosis structure:

```
* Subject
  * diagnosis
    (Diagnosis record 1)
      * disease_diagnosis: "rheumatoid arthritis"
      * disease_length: "20 years"
    (Diagnosis record 2)
      * disease_diagnosis: "pancreatic ductal adenocarcinoma"
      * disease_length: "6 months"
```

A query of `disease_diagnosis = pancreatic ductal adenocarcinom` and `disease_length > 10` will result in the above Subject being returned, even though the subject has not had pancreatic ductal adenocarcinom for more than 10 years. This is because each of the predicates in the query are true given the above subject. That is the subject has a `disease_diagnosis = pancreatic ductal adenocarcinom` and a `disease_length > 10`. It is currently not possible to perform the above query using the current implementation of the ADC API.

This query would only result in the desired outcome if and only if there was one disease record for the subject as given below.

```
* Subject
  * diagnosis
```

(continues on next page)

(continued from previous page)

```
  (Diagnosis record 1)
    * disease_diagnosis: "pancreatic ductal adenocarcinoma"
    * disease_length: "20 years"
```

If there is more than one diagnosis, it is necessary to search for one of the criteria (e.g. `disease_diagnosis = pancreatic ductal adenocarcinom`), download the resulting data, and determine if the other criteria is true for that disease record for that subject.

A planned extension to solve this issue is being devloped.

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.

Other endpoint query fields

Unlike repertoire data, data repositories are expected to store billions of other records (e.g. Rearrangement, CellExpression, Clone, Cell), 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.

Minimal Rearrangement Query Fields

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.

Minimal Clone Query Fields

Field(s)	Description
clone_id, repertoire_id, data_processing_id	Identifiers; clone_id allows for query of that specific clone object in the repository, while repertoire_id and data_processing_id are links to the repertoire metadata for the clone.
v_call, d_call, j_call, junction_aa, junction_aa_length, clone_count	Commonly used clone annotations.

Minimal Cell Query Fields

Field(s)	Description
cell_id, repertoire_id, data_processing_id	Identifiers; cell_id allows for query of that specific cell object in the repository, while repertoire_id and data_processing_id are links to the repertoire metadata for the cell.
rearrangements, receptors, reactivity_measurements, expression_study_method, virtual_pairing	Commonly used cell attributes.

Minimal CellExpression Query Fields

Field(s)	Description
expression_id, cell_id, repertoire_id, data_processing_id	Identifiers; expression_id allows for query of that specific expression object in the repository, while repertoire_id and data_processing_id are links to the repertoire metadata for the cell. cell_id allows for searching for expression data for a specific cell.
property, value	Commonly used cell expression attributes.

Minimal Receptor Query Fields

Field(s)	Description
receptor_id, receptor_hash	Identifiers; receptor_id allows for query of that specific receptor object in the repository. receptor_hash allows for a fast/efficient look up of the globally unique receptor hash.
receptor_type, receptor_variable_domain_1_aa, receptor_variable_domain_1_locus, receptor_variable_domain_2_aa, receptor_variable_domain_2_locus, receptor_ref, reactivity_measurements	Commonly used receptor attributes.

Minimal ReceptorReactivity Query Fields

Field(s)	Description
receptor_reactivity_id	Identifiers; receptor_reactivity_id allows for query of that specific receptor reactivity object in the repository.
ligand_type, antigen_type, antigen, antigen_source_species, peptide_sequence_aa, mhc_class, mhc_gene_1, mhc_allele_1, mhc_gene_2, mhc_allele_2, reactivity_method, reactivity_readout, reactivity_value, reactivity_unit	Commonly used receptor reactivity attributes.

Data repository specific limits

A data repository may impose limits on the size of the data sent to and returned by the repository. This might be because of limitations imposed by the back-end database being used or because of the need to manage the load placed on the server. For example, MongoDB databases have document size limits (16 megabytes) which limit the size of a query that can be sent to a repository and the size of a single repertoire or rearrangement object that is returned. As a result a repository might choose to set a maximum query size.

Size limits can be retrieved from the `info` endpoint. These limits are repository wide (the same for all endpoints). If the data repository does not provide a limit, then no limit is assumed.

Field	Description
`max_siz`	The maximum value for the `size` query parameter. Attempting to retrieve data from a repository beyond this maximum should trigger an error response. The error response should include information about why the query failed and what the maximum size limit is.
`max_que`	The maximum size (in bytes) of the JSON query object. Atempting to send a query to a repository larger than this size should trigger an error.

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.

Asynchronous API

The ADC API is a synchronous API, which means that a query is executed immediately and all data satisfying the query is returned in the response, regardless of the length of time to execute the query and the amount of data returned in the response. The network connection between the client and server is maintained until the request is complete.

Repositories may wish to have better control over queries that either take a long time to run or which return large amounts of data. Partial control is provided by utilizing the *max_size* limit, which restricts the amount of data returned; however, there is nothing to restrict the number of concurrent queries nor how long an individual query can run. For network services with time restrictions, this can cause timeouts as the client/server connection is cut if queries take too long.

The ADC Asynchronous API is an extension that repositories may implement which provides control over query execution. Queries are performed asynchronously, which means that incoming query requests return immediately with a status record, queries are executed in the future by the repository, and the status record is updated when the query is complete and the data is ready.

Currently only the `rearrangement` endpoint is required, but other endpoints may be added in future versions. The Async API mirrors the ADC API with the query parameters that are accepted with the exception of `facets` (this may be added in future versions).

Repositories should assign a unique `query_id` for every incoming query request, which will allow the status of the query to be tracked. Requests may specify a URL for notifications as the query status changes; however, notifications do not have guaranteed delivery so client programs should utilize polling as a backup. Repositories should update the query status and send out notifications, as the query goes through various processing stages. The current statuses that repositories must support are listed below.

- PENDING: A query which has been received but not yet completely accepted for submission. The repository may perform additional error checks before accepting the query.

- SUBMITTED: A query that has passed any initial error checks and is submitted to be executed.
- PROCESSING: A query that is currently being processed.
- FINISHED: A query that has finished and data is available.
- ERROR: A query where an error occurred that prevents completion.
- EXPIRED: A query that has finished but has expired and the data is no longer available.

When a query is FINISHED, the data should be made available with a download URL. The download does not have to be through the same API service, which allows repositories to utilize services that are more efficient at downloading large data files.

Async API Endpoints

The ADC Async API is versioned with the major version number (`v1`) as part of the base path (`/airr/async/v1`) for all endpoints. Each endpoint provides 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/rearrangement`	Query rearrangements asynchronously	`POST`	Upon success, returns a query status object with a unique `query_id`.
`/v1/status/{query_id}`	Retrieve query status given its `query_id`	`GET`	Upon success, returns the query status information.

Request Asynchronous Query on Rearrangements

The `rearrangement` endpoint provides access to all fields in the *Rearrangement schema*. Unlike the ADC API which limits the set of queryable fields, any field can be included in an asynchronous query.

This example queries for rearrangements with a specific junction amino acid sequence among a set of repertoires. This is the exact same query that can be sent to the synchronous ADC API.

```
curl --data @query1_rearrangement.json -H 'content-type: application/json' https://
↪vdjserver.org/airr/async/v1/rearrangement
```

The content of the `JSON payload`.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":[
                        "2603354229190496746-242ac113-0001-012",
                        "2618085967015776746-242ac113-0001-012",
                        "2633633748627296746-242ac113-0001-012",
                        "2564613624180576746-242ac113-0001-012"
                    ]
```

(continues on next page)

(continued from previous page)

```
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
            }
        ]
    },
    "fields":["repertoire_id","sequence_id","v_call","productive"],
    "format":"tsv"
}
```

Here is an example response with the unique `query_id`.

```
{
  "message":"rearrangement lrq submitted.",
  "query_id":"1978706891589873170-242ac118-0001-012"
}
```

Retrieve Status of an Asynchronous Query

The status of an asynchronous query can be retrieved at any time by sending a request with the `query_id`. Here is an example using the `query_id` from the above request.

```
curl https://vdjserver.org/airr/async/v1/status/1978706891589873170-242ac118-0001-012
```

Here is an example response:

```
{
  "query_id":"1978706891589873170-242ac118-0001-012",
  "endpoint":"rearrangement",
  "status":"FINISHED",
  "message":null,
  "created":"2024-02-25T16:19:07.801-06:00",
  "estimated_count":10,
  "final_file":"1978706891589873170-242ac118-0001-012.airr.tsv",
  "download_url":"https://vdj-agave-api.tacc.utexas.edu/postits/v2/f8d52cdb-9234-41e3-
↪a908-95c9034a3360-010"
}
```

Retrieve Data for an Asynchronous Query

When a query is FINISHED, the `download_url` field will provide the URL for retrieving the data. The `final_file` field is a repository assigned file name, usually made unique in some way by using the `query_id`, that is provided for convenience, but clients are allowed to use whatever filename they wish. Here is an example to download the data for above query.

```
curl -o 1978706891589873170-242ac118-0001-012.airr.tsv https://vdj-agave-api.tacc.utexas.
↪edu/postits/v2/f8d52cdb-9234-41e3-a908-95c9034a3360-010
```

Request Notifications for Query Status Changes

The client can request that a URL be called by the repository whenever the status of a query changes. This can only be done with the initial query request. A notification object is provided with the initial query request that specifies the URL to be called, whether the GET or POST method is used, and what status changes to receive. This object is provided with the `notification` parameter. Here is the above example query request where a notification URL has been added.

```
{
    "filters":{
        "op":"and",
        "content": [
            {
                "op":"in",
                "content": {
                    "field":"repertoire_id",
                    "value":[
                        "2603354229190496746-242ac113-0001-012",
                        "2618085967015776746-242ac113-0001-012",
                        "2633633748627296746-242ac113-0001-012",
                        "2564613624180576746-242ac113-0001-012"
                    ]
                }
            },
            {
                "op":"=",
                "content": {
                    "field":"junction_aa",
                    "value":"CARDPRSYHAFDIW"
                }
            }
        ]
    },
    "fields":["repertoire_id","sequence_id","v_call","productive"],
    "format":"tsv",
    "notification":{
        "url"": "https://notify.example.com",
        "method": "POST"
    }
}
```

When sending a notification, the repository will call the URL using requested method and provide data with information about the query status change. This data is exactly the same as what is returned by the `status` endpoint.

2.7 AIRR Software WG - Guidance for AIRR Software Tools

Version 1.0

2.7.1 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:

Brief description 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 Standards* 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	

2.7.2 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.
ImmuneDB	0.29.10	Input	Rosenfeld et al. Front Immunol, 2018,
Scirpy	0.4.2	Input	Sturm et al. Bioinformatics, 2020,
Immcantation Framework	4	Input, Output	Vander Heiden et al. Bioinformatics, 2014,; Gupta et al., Bioinformatics, 2015,
CompAIRR	1.3.1	Input	Rognes T, Scheffer L, Greiff V, Sandve GK, BioRxiv, 2021 (preprint),
ImmuneML	2.0.4	Input, Output	Pavlovic et al., BioRxiv, 2021 (preprint),
Dandelion	0.1.10	Output	Suo, C., Polanski, K., Dann, E. et al., Nature Biotechnology, 2023,
TRUST4	1.0.8	Output	Song, L., Cohen, D., Ouyang, Z. et al., Nature Methods, 2021,
partis	0.16.0	Input, Output	Ralph D and Matsen F IV, PLOS Computational Biology, 2022,
nf-core/airrflow	3.2.0	Input, Output	Gisela Gabernet, Susanna Marquez, Alexander Peltzer, et al, 2023,
AnalyzAIRR	1.1.0	Input	V. Mhanna, G. Pires, G. Bohl, et al., AnalyzAIRR,

2.7.3 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.

2.7.4 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 Standards* 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.

2.7.5 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.

2.7.6 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.

2.7.7 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.

2.8 AIRR Ontologies and Vocabularies Sub-WG

2.8.1 Summary

The "Ontologies and Vocabularies Team" was initial formed as a joint interest group of the Common Repository (ComRepo) and the Minimal Standards (MiniStd) working groups (WG) of the AIRR Community. When the two WG merged

into the current Standards WG in December 2020, OntoVoc became a Sub-WG of it. The long-term aim of the Sub-WG is to define standard vocabularies and ontologies to be used by AIRR-compliant repositories.

2.8.2 Ontology Data Representation

The nodes in an ontology 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. Ontology entities in the AIRR Data Standard reflect this model, with each AIRR field that is represented as an ontology recorded with a global *ontology ID* (`id`) and the corresponding *label* (`label`).

Within the AIRR Standards, Compact URIs (CURIEs) are used to represent *ontology IDs* or *persistent IDs*. CURIEs are a standardized way to abbreviate International Resource Identifiers (IRI, [**?**]), which include URIs and URLs as subsets. 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 also used by IEDB databases to reduce redundancies (mainly in the leading part of IRIs).

For example, a typical CURIE would 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. Although common, it is not always the case that a resolved CURIE (the IRI *prefix* plus the *local ID*) can be used as a URL directly to look up the CURIE using a web browser.

The AIRR Schema provides a `CURIEMap`, a list of AIRR approved CURIE *prefixes* along with a `map` of at least one `iri_prefix` (i.e., a replacement string to construct the complete IRI) for each *prefix*. As the `iri_prefix` might differ between *provider*-specific implementations of an ontology (e.g., NCBI Taxonomy), the `CURIEMap` supports multiple `iri_prefix` entries for a given *prefix*. Finally, the `CURIEMap` should also provide a default `map` and `provider` for each *prefix*. Complementary to this, the `InformationProvider` list describes the mechanism to computationally look up a resolved IRI (e.g., the `iri_prefix` and the *local ID*) by specifying how to make a request to the *provider* as well as describing the format in which the request response will be provided.

The `CURIEMap` serves several 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 are prohibited. This does not affect the ability of repositories to use such custom prefixes internally.
2. It simplifies resolution of CURIEs. The `iri_prefix` lists for each *prefix* should not be considered to be exhaustive. However, when using a custom `iri_prefix`, it must be ensured that the expanded IRI still refers to the same concept/instance as when using the default `iri_prefix`.
3. It simplifies computation using CURIES. It is possible to use the `provider` for a *prefix* as a mechanism to look up a CURIE from a *provider* with a defined response (See below)

It should be explicitly noted that the `CURIEMap` 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.

2.8.3 General Policies

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
7. SHOULD comply to the OBO Foundry Principles. This does not imply a preference.

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.
- ad *(7)*: This is an endorsement of the OBO Foundry *Principles*, not of the OBO Foundry *Ontologies* in general. Hence, also non-OBO have an equal standing if they comply to the Principles.

2.8.4 Approved Ontologies

- Cell ontology (CL)
  - used in:
    - Cell subset (`cell_subset`, Tissue and Cell Processing)
  - CURIE summary
    - CURIE Prefix: `CL`
    - CURIE IRI Prefix: `http://purl.obolibrary.org/obo/CL_`
  - example AIRR use
    - “cell_subset.id” : “CL:0000542”
    - “cell_subset.label” : “lymphocyte”
  - 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*)
  - CURIE summary
    - CURIE Prefix: `DOID`
    - CURIE IRI Prefix: `http://purl.obolibrary.org/obo/DOID_`

  - example AIRR use
    - “disease_diagnosis.id” : “DOID:9538”
    - “disease_diagnosis.label” : “multiple myeloma”
  - 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)
  - CURIE summary
    - CURIE Prefix: `NCBITAXON`
    - CURIE IRI Prefixes: `http://purl.obolibrary.org/obo/NCBITaxon_`, `http://purl.bioontology.org/ontology/NCBITAXON/`
  - example AIRR use
    - “species.id” : “NCBITAXON:9606”
    - “species.label” : “Homo sapiens”
  - 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*)
  - CURIE summary
    - CURIE Prefix: `NCIT`

        * CURIE IRI Prefixes: `http://purl.obolibrary.org/obo/NCIT_`, `http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#`
    - example AIRR use
        * "study_type.id" : "NCIT:C15197"
        * "study_type.label" : "Case-Control 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*)
    - CURIE summary
        * CURIE Prefix: `UO`
        * CURIE IRI Prefix: `http://purl.obolibrary.org/obo/UO_`
    - example AIRR use
        * "age_unit.id" : "UO:0000036"
        * "age_unit.label" : "year"
    - 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*)
    - CURIE summary
        * CURIE Prefix: `UBERON`
        * CURIE IRI Prefix: `http://purl.obolibrary.org/obo/UBERON_`

  - example AIRR use
    - "tissue.id" : "UBERON:0002371"
    - "tissue.label" : "bone marrow"
  - 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)

2.8.5 Computing with Ontologies

One of the key goals of using ontologies is to enable analysis tools to perform computation using the information in those ontologies. The AIRR Schema's `CURIEMap` lists one or more *providers* for each CURIE *prefix* that can be used programmatically by analysis tools. Although the AIRR Schema lists multiple providers for each ontology, this section focuses on the use of the EBI OLS *provider's* OLS Web API interface for querying ontologies.

If we consider the `DOID` *prefix* from the `CURIEMap`, the section below defines the use of the Human Disease Ontology (DOID) within the AIRR Standard:

```
DOID:
  type: ontology
  default:
    map: OBO
    provider: OLS
  map:
    OBO:
      iri_prefix: "http://purl.obolibrary.org/obo/DOID_"
```

We see that the default `map` for DOID is `OBO` map, and the OBO map's `iri_prefix` is `http://purl.obolibrary.org/obo/DOID_`. Thus the mapping of the CURIE `DOID:9538` (the CURIE for disease "multiple myeloma") will yield the resolved string `http://purl.obolibrary.org/obo/DOID_9538`. By the strictest of defintions, this is a valid IRI and should only be considered an identifier, but in this case this IRI is also a URL and can be used to look up the CURIE.

If we consider the default DOID `provider` in the `CURIEMap`, we see that it is `OLS`. Then, in the `InformationProvider` object of the AIRR Schema, under `provider` we see:

```
InformationProvider:
  provider:
    OLS:
      request:
        url: "https://www.ebi.ac.uk/ols/api/ontologies/{ontology_id}/terms?iri={iri}"
        response: application/json
```

And later we see that the `parameters` for `OLS` are:

```
parameter:
  CL:
    Ontobee:
      ontology_id: CL
    OLS:
      ontology_id: cl
  DOID:
    Ontobee:
      ontology_id: DOID
    OLS:
      ontology_id: doid
```

The above tells us that we can use the OLS `provider` to look up ontology terms. The `{iri}` component of the `url` string tells us that we need to use the resolved IRI and the `{ontology_id}` component tells us that we need to replace the `ontology_id` parameter in the URL with the DOID OLS parameter in the specification, which is the string `doid`. Thus the fully resolved URL to query for the CURIE `DOID:9538` would be:

```
https://www.ebi.ac.uk/ols/api/ontologies/doid/terms?iri=http://purl.obolibrary.org/obo/
↪DOID_9538
```

Again, referring to the OLS `provider` we see that we can expect an `application/json` response to the above query, and indeed the response we receive from the above starts with a JSON object as follows.

```
{
 "_embedded" : {
   "terms" : [ {
     "iri" : "http://purl.obolibrary.org/obo/DOID_9538",
     "label" : "multiple myeloma",
     "description" : [ "A myeloid neoplasm that is located_in the plasma cells in bone␣
↪marrow." ],
     "annotation" : {
       "comment" : [ "OMIM mapping confirmed by DO. [SN]." ],
       "database_cross_reference" : [ "ICD10CM:C90.0", "MESH:D009101", "ICD9CM:203.0",
↪"GARD:7108", "NCI:C3242", "OMIM:254500", "ORDO:29073", "EFO:0001378", "SNOMEDCT_US_
↪2020_09_01:94705007", "UMLS_CUI:C0026764" ],
       "has_obo_namespace" : [ "disease_ontology" ],
       "id" : [ "DOID:9538" ]
     },
     "synonyms" : [ "plasma cell myeloma" ],
     "ontology_name" : "doid",
     "ontology_prefix" : "DOID",
     "ontology_iri" : "http://purl.obolibrary.org/obo/doid.owl",
     "is_obsolete" : false,
     "term_replaced_by" : null,
     "is_defining_ontology" : true,
     "has_children" : true,
     "is_root" : false,
     "short_form" : "DOID_9538",
     "obo_id" : "DOID:9538",
     [Content edited because of length]
```

In this repsonse, you can see that the Ontology object that we requested has a `label` field that contains the value `multiple myeloma` and that the `id` field has a value of `DOID:9538`.

It is beyond the scope of this document to describe in detail the JSON structure of each of the providers, but this information can be discovered through the `provider` web sites. It should be noted that all Ontology objects in the AIRR specification have the OLS as a `provider` and therefore the method above can be used for any of the ontologies in the AIRR specification. Please see the OLS Web API documentation for details of the JSON response for the OLS `provider`.

2.9 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.

2.9.1 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:

1. Submit sequence and data to Genbank or ENA, following the Genbank/ENA workflow.
2. Submit the inferred sequences to IARC via OGRDB, following the OGRDB Submission Guide.

Additional information is available at the OGRDB Website.

References

2.10 Community Resources

2.10.1 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 (or higher) 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.
Alakazam	1.0.1	Input and output	Gupta & Vander Heiden et al. Bioinformatics, 2015.
AnalyzAIRR	1.2.2	Input	Mhanna et al. ImmunoInformatics, 2025. <https://doi.org/10.1016/j.immuno.2025.100052>`_
Cell Ranger	4.0	Output	10x Genomics, Inc. Pleasanton, CA USA.
Change-O	0.4.2	Input, output and conversion	Gupta & Vander Heiden et al. Bioinformatics, 2015.
compAIRR	0.2.0	Input	Rognes et al. Bioinformatics, 2022.
Decombinator	4.0.1	Output	Oakes et al. Front Immunol, 2017.
dowser	0.1.0	Input	Hoehn et al. PLoS Comput Biol, 2022.
IMGT/HighV-QUEST	1.7.0	Output	Alamyar et al. Methods Mol Biol, 2012.
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.
immunarch	0.6.5	Input	ImmunoMind Team. 2019
ImmuneDB	0.24.0	Output	Rosenfeld et al. Front Immunol, 2018.
immuneML	2.0.0	Input	Pavlovic et al. Nat Mach Intell, 2021.
immuneSIM	0.8.7	Output	Weber et al. Bioinformatics, 2020.
iReceptor	3.0	Input and output	Corrie et al. Immunol Rev, 2018.
MiXCR	4.0	Output	Bolotin et al. Nat Methods, 2015.
RAbHIT	0.1.5	Input and output	Gidoni et al. Nat Commun, 2019.
scirpy	0.7	Input and output	Sturm et al. Bioinformatics, 2020.
SCOPer	1.0.1	Input and output	Nouri & Kleinstein. Bioinformatics, 2018.
SHazaM	1.0.0	Input and output	Gupta & Vander Heiden et al. Bioinformatics, 2015.
SONAR	3	Output	Schramm et al. Front Immunol, 2016.
sumrep	1.0	Input	Olson et al. Front Immunol, 2019.
TIgGER	1.0.0	Input and output	Gadala-Maria et al. PNAS, 2015.
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.

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

The repositories that are part of the ADC are listed on the AIRR Community ADC Registry github site. In order to find data across all of the repositories in the ADC, it is necessary to query all of these repositories.

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
- VDJServer Community Data Portal

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

AIRR Data Commons Repositories

A full list of AIRR Compliant repositories that implement the AIRR Data Commons API can be found at the AIRR Community ADC Registry github site.

2.10.2 Useful Websites for the AIRR Community

- The AIRR Community
- B-T.CR Forum
- The AIRR Community GitHub
- The AIRR Standards GitHub Repository
- The AIRR Community Docker Hub

2.11 Appendix A: Key Terms

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

Table 4: Definitions and abbreviation

Term	Abbreviation	Retired synonym	Definition
Adaptive Immune Receptor Repertoire	AIRR		A collection of vertebrate immune receptors whose loci typically undergo RAG-mediated rearrangement, i.e., immunoglobulins and T cell receptors.
AIRR Community	AIRR C		The Adaptive Immune Receptor Repertoire Community is a grassroots collaboration for developing and promoting best practices for generating, analyzing, curating, and sharing AIRR-seq data, see [DOI:10.3389/fimmu.2017.01418]. As of 12/2017, AIRR-C became a Committee within The Antibody Society (TAbS) and is therefore also known as "The AIRR Community Committee within The Antibody Society".
AIRR Community Standards	AIRR C Standards	AIRR Standards	The collection of all norms, including standards, guidelines, best practices, recommendations and definitions, which have been developed and published by the AIRR Community, in their respective current version.
AIRR Data Commons	ADC		The network of data repositories, which make AIRR-seq data FAIR by implementing the ADC API, adhering to the MiAIRR metadata standard and provide data in an AIRR Format.
AIRR Data Commons Application Program Interface	ADC API		The standard defining how AIRR-seq data located in a repository can be programmatically searched and retrieved.
AIRR Data Schema			The formal definition of all specified properties of an AIRR-seq study. This includes groupings of properties, relations between properties and the permissible representations of property values.
AIRR Formats		AIRR Data Representation	The collection of definitions of on-disk and/or in-transit representation of the entities specified in the AIRR Data Schema and in MiAIRR.
AIRR Sequencing	AIRR seq		The experimental observation of an Adaptive Immune Receptor Repertoire by means of nucleic acid sequencing.
API			Application Programming Interface
CAIRR			CEDAR AIRR
CEDAR			Center for Expanded Data Annotation and Retrieval
Clone			1) [biological definition] A group of B or T cells that are all descended from the same naive ancestal cell and carry the same rearrangements. In B cells, the Ig sequence may vary between members of a Clone due to somatic hypermutation. To indicate this fact, the term "clonal linage" is preferred by some researchers. 2) [informatic

2.12 References

BIBLIOGRAPHY

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

[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

Symbols

A