

# Package ‘immunotation’

December 11, 2024

**Type** Package

**Title** Tools for working with diverse immune genes

**Version** 1.15.0

**Date** 2021-08-09

## **Description**

MHC (major histocompatibility complex) molecules are cell surface complexes that present antigens to T cells. The repertoire of antigens presented in a given genetic background largely depends on the sequence of the encoded MHC molecules, and thus, in humans, on the highly variable HLA (human leukocyte antigen) genes of the hyperpolymorphic HLA locus. More than 28,000 different HLA alleles have been reported, with significant differences in allele frequencies between human populations worldwide. Reproducible and consistent annotation of HLA alleles in large-scale bioinformatics workflows remains challenging, because the available reference databases and software tools often use different HLA naming schemes. The package immunotation provides tools for consistent annotation of HLA genes in typical immunoinformatics workflows such as for example the prediction of MHC-presented peptides in different human donors. Converter functions that provide mappings between different HLA naming schemes are based on the MHC restriction ontology (MRO). The package also provides automated access to HLA allele frequencies in worldwide human reference populations stored in the Allele Frequency Net Database.

**License** GPL-3

**Encoding** UTF-8

**LazyData** false

**Collate** 'external\_resources\_input.R' 'AFND\_interface.R'  
'MRO\_interface\_helper.R' 'MRO\_interface.R' 'MACUI\_interface.R'  
'nomenclature\_queries.R' 'AFND\_queries.R' 'visualization.R'

**Depends** R (>= 4.1)

**Imports** stringr, ontologyIndex, curl, ggplot2, readr, rvest, tidyr,  
xml2, maps, rlang

**Suggests** BiocGenerics, rmarkdown, BiocStyle, knitr, testthat, DT

**VignetteBuilder** knitr

**RoxygenNote** 7.1.1

**biocViews** Software, ImmunoOncology, BiomedicalInformatics, Genetics, Annotation

**BugReports** <https://github.com/imkeller/immunotatation/issues>

**git\_url** <https://git.bioconductor.org/packages/immunotatation>

**git\_branch** devel

**git\_last\_commit** 6821974

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.21

**Date/Publication** 2024-12-10

**Author** Katharina Imkeller [cre, aut]

**Maintainer** Katharina Imkeller <k.imkeller@dkfz.de>

## Contents

assemble_protein_complex . . . . .	3
build_allele_group . . . . .	4
check_geographics . . . . .	4
check_hla_locus . . . . .	5
check_hla_selection . . . . .	5
check_population . . . . .	6
check_sample_size . . . . .	6
check_standard . . . . .	7
create_encode_handle . . . . .	7
decode_MAC . . . . .	8
encode_MAC . . . . .	8
extract_population_id . . . . .	9
extract_population_info . . . . .	9
extract_population_name . . . . .	10
extract_sample_info . . . . .	10
fetch_encoded_MAC . . . . .	11
getURL . . . . .	11
get_external_file . . . . .	12
get_G_group . . . . .	12
get_mhcpan_input . . . . .	13
get_nb_pages . . . . .	14
get_P_group . . . . .	14
get_serotypes . . . . .	15
get_valid_geographics . . . . .	15
get_valid_organisms . . . . .	16
human_protein_complex_table . . . . .	16
load_mro . . . . .	17
mro.obo . . . . .	17
parse_allele_freq_html . . . . .	17
parse_haplotype_freq_html . . . . .	18

<i>assemble_protein_complex</i>	3
plot_allele_frequency . . . . .	18
query_allele_frequencies . . . . .	19
query_haplotype_frequencies . . . . .	20
query_population_detail . . . . .	21
read_complete_freq_table . . . . .	22
read_population_detail . . . . .	22
retrieve_chain_lookup_table . . . . .	23
verify_parameters . . . . .	23
<b>Index</b>	<b>25</b>

---

`assemble_protein_complex`  
*Assemble protein complex*

---

### Description

Assemble a table of MHC protein complexes for a given organism.

### Usage

```
assemble_protein_complex(organism)
```

### Arguments

<code>organism</code>	Organism for which the lookup should be built (e.g. "human", "mouse", ...). The list of valid organisms can be found using the function <code>get_valid_organisms</code>
-----------------------	--

### Value

a data frame with the MHC complexes annotated in MRO (only completely annotated complexes are returned)

### Examples

```
assemble_protein_complex(organism = "mouse")
```



---

check\_hla\_locus      *check\_hla\_locus, stops if input not adequate*

---

**Description**

check\_hla\_locus, stops if input not adequate

**Usage**

check\_hla\_locus(hla\_locus)

**Arguments**

hla\_locus      locus of hla frequencies to query

**Value**

TRUE

---

check\_hla\_selection      *check\_hla\_selection, stops if input not adequate*

---

**Description**

check\_hla\_selection, stops if input not adequate

**Usage**

check\_hla\_selection(hla\_selection, query\_type)

**Arguments**

hla\_selection      HLA alleles used for selection  
query\_type      ["allele"|"haplotype"] type of AFND query

**Value**

TRUE

---

check_population	<i>check_population, stops if input not adequate</i>
------------------	--

---

**Description**

check\_population, stops if input not adequate

**Usage**

```
check_population(hla_population)
```

**Arguments**

hla\_population population id

**Value**

TRUE

---

check_sample_size	<i>check_sample_size, stops if input not adequate</i>
-------------------	---

---

**Description**

check\_sample\_size, stops if input not adequate

**Usage**

```
check_sample_size(hla_sample_size_pattern, hla_sample_size)
```

**Arguments**

hla\_sample\_size\_pattern  
 one of "bigger\_than", "equal", "less\_than", "less\_equal\_than", "bigger\_equal\_than", "different"

hla\_sample\_size  
 integer number used for population size

**Value**

TRUE

---

check\_standard      *check\_standard, stops if input not adequate*

---

**Description**

check\_standard, stops if input not adequate

**Usage**

check\_standard(standard)

**Arguments**

standard      one of "a" - all, "s" - silver, "g" - gold

**Value**

TRUE

---

create\_encode\_handle      *create\_encode\_handle*

---

**Description**

create\_encode\_handle

**Usage**

create\_encode\_handle(allele\_names)

**Arguments**

allele\_names      list of HLA alleles

**Value**

curl handle

---

decode_MAC	<i>Decode MAC</i>
------------	-------------------

---

### Description

Decode a multiple allele code (MAC) into a list of HLA alleles. #' The National Marrow Donor Program (NMDP) uses [MAC](<https://bioinformatics.bethematchclinical.org/hla-resources/allele-codes/allele-code-lists/>) to facilitate the reporting and comparison of HLA alleles. MAC represent groups of HLA alleles and are useful when the HLA typing is ambiguous and does not allow to narrow down one single allele from a list of alleles.

### Usage

```
decode_MAC(MAC)
```

### Arguments

MAC	multiple allele code (e.g. "A*01:ATJNV")
-----	--

### Value

list of HLA alleles

### Examples

```
MAC <- "A*01:ATJNV"
decode_MAC(MAC)
```

---

encode_MAC	<i>Encode MAC</i>
------------	-------------------

---

### Description

Encode a list of HLA alleles into multiple allele code (MAC). The National Marrow Donor Program (NMDP) uses [MAC](<https://bioinformatics.bethematchclinical.org/hla-resources/allele-codes/allele-code-lists/>) to facilitate the reporting and comparison of HLA alleles. MAC represent groups of HLA alleles and are useful when the HLA typing is ambiguous and does not allow to narrow down one single allele from a list of alleles.

### Usage

```
encode_MAC(allele_list)
```

### Arguments

allele_list	list of HLA alleles (e.g. c("A*01:01:01", "A*02:01:01", "A*03:01"))
-------------	---

**Value**

encoded MAC

**Examples**

```
allele_list <- c("A*01:01:01", "A*02:01:01", "A*03:01")
encode_MAC(allele_list)
```

---

*extract\_population\_id extract\_population\_id*

---

**Description**

extract the population ids from the html result

**Usage**

```
extract_population_id(data)
```

**Arguments**

data                   html from AFND website

**Value**

population ids

---

*extract\_population\_info*  
*extract\_population\_info*

---

**Description**

extract\_population\_info

**Usage**

```
extract_population_info(data)
```

**Arguments**

data                   html input page

**Value**

population information

---

extract\_population\_name  
*extract\_population\_name*

---

**Description**

extract\_population\_name

**Usage**

extract\_population\_name(data)

**Arguments**

data           html input page

**Value**

population name

---

extract\_sample\_info   *extract\_sample\_info*

---

**Description**

extract\_sample\_info

**Usage**

extract\_sample\_info(data)

**Arguments**

data           html input page

**Value**

sample information

---

fetch_encoded_MAC	<i>fetch_encoded_MAC</i>
-------------------	--------------------------

---

**Description**

fetch\_encoded\_MAC

**Usage**

fetch\_encoded\_MAC(handle)

**Arguments**

handle	curl handle
--------	-------------

**Value**

curl handle fetch

---

getURL	<i>getURL</i>
--------	---------------

---

**Description**

getURL

**Usage**

```
getURL(
  URL,
  N.TRIES = 2L,
  read_method = c("delim", "lines", "html"),
  skip = 0,
  delim = "\\t",
  col_names = TRUE
)
```

**Arguments**

URL	Indicated the url that will be read
N.TRIES	Integer, how often should the function try to read the URL?
read_method	Method to be used for reading of URL content ("delim" -> readr::read_delim, "lines" -> readr::read_lines, "html" -> xml2::read_html)
skip	integer indicating how many lines to skip when reading URL
delim	pattern used for delim (passed to delim of read functions)
col_names	list of colnames to use

**Value**

returns a the content of the URL. The format of the return object depends on the read\_method that was used.

---

get_external_file	<i>get_external_file</i>
-------------------	--------------------------

---

**Description**

get\_external\_file

**Usage**

```
get_external_file(file, skip = 0, delim = "\t", col_names = TRUE)
```

**Arguments**

file	Indicated the file that will be read
skip	integer indicating how many lines to skip when reading URL
delim	pattern used for delim (passed to delim of read functions)
col_names	list of colnames to use

**Value**

returns a the content of the file. The format of the return object depends on the read\_method that was used.

---

get_G_group	<i>G groups</i>
-------------	-----------------

---

**Description**

Get the G groups for a list of HLA alleles. [G groups]([http://hla.alleles.org/alleles/g\\_groups.html](http://hla.alleles.org/alleles/g_groups.html)) are groups of HLA alleles that have identical nucleotide sequences across the exons encoding the peptide binding domains.

**Usage**

```
get_G_group(allele_list)
```

**Arguments**

allele_list	List of alleles.
-------------	------------------

**Value**

Named list of G-groups the input alleles belong to.

**Examples**

```
allele_list <- c("DQB1*02:02:01", "DQB1*06:09:01")
get_G_group(allele_list)
```

---

<code>get_mhcpan_input</code>	<i>Get format for NetMHCpan tools</i>
-------------------------------	---------------------------------------

---

**Description**

NetMHCpan tools for MHC-peptide binding prediction require HLA complex names in a specific format. `get_mhcpan_input` formats a list of HLA alleles into a list of NetMHC-formated complexes.

**Usage**

```
get_mhcpan_input(allele_list, mhc_class)
```

**Arguments**

<code>allele_list</code>	list of HLA alleles (e.g. <code>c("A*01:01:01", "B*27:01")</code> )
<code>mhc_class</code>	<code>["MHC-I" "MHC-II"]</code> indicated which NetMHC you want to use.

**Value**

protein chain list as formatted for MHCpan input

**Examples**

```
allele_list <- c("A*01:01:01", "B*27:01")
get_mhcpan_input(allele_list, mhc_class = "MHC-I")
```

---

get_nb_pages	<i>get_nb_pages</i>
--------------	---------------------

---

**Description**

get\_nb\_pages

**Usage**

```
get_nb_pages(page_tbl)
```

**Arguments**

page\_tbl      html input page

**Value**

integer number of pages in the html input

---

get_P_group	<i>P groups</i>
-------------	-----------------

---

**Description**

Get the P groups for a list of HLA alleles. [P groups]([http://hla.alleles.org/alleles/p\\_groups.html](http://hla.alleles.org/alleles/p_groups.html)) are groups of HLA alleles that have identical protein sequences in the peptide binding domains.

**Usage**

```
get_P_group(allele_list)
```

**Arguments**

allele\_list    list of HLA alleles

**Value**

Named list of P-groups the input alleles belong to.

**Examples**

```
allele_list <- c("DQB1*02:02:01", "DQB1*06:09:01")
get_P_group(allele_list)
```

---

get_serotypes	<i>Serotypes</i>
---------------	------------------

---

**Description**

Get the serotypes of the MHC complexes encoded by a list of MHC alleles.

**Usage**

```
get_serotypes(allele_list, organism = "human", mhc_type)
```

**Arguments**

allele_list	List of allele
organism	Organism to be used for MRO lookup. If the organism does not match the given allele, a empty object is returned.
mhc_type	["MHC-I" or "MHC-II"] MHC class to use for MRO lookup.

**Value**

Named list of serotypes, which only contains complexes contained in the MRO. If no serotype is annotated for a given complex, the list element is NA.

**Examples**

```
allele_list <- c("A*01:01:01", "B*27:01")
get_serotypes(allele_list, mhc_type = "MHC-I")
```

---

get_valid_geographics	<i>get_valid_geographics</i>
-----------------------	------------------------------

---

**Description**

get a list of valid countries, regions and ethnic origins

**Usage**

```
get_valid_geographics()
```

**Value**

list of valid countries, regions and ethnic origin

---

```
get_valid_organisms  get_valid_organisms
```

---

**Description**

get the list of organisms that are part of the MRO annotation

**Usage**

```
get_valid_organisms()
```

**Value**

list of organisms

**Examples**

```
get_valid_organisms()
```

---

```
human_protein_complex_table  
      human_protein_complex_table
```

---

**Description**

human\_protein\_complex\_table

**Usage**

```
human_protein_complex_table
```

**Format**

An object of class `data.frame` with 12385 rows and 8 columns.

**Details**

human\_protein\_complex\_table: human\_protein\_complex\_table.

**Examples**

```
# The human protein complex table is available in the following  
# exported variable  
human_protein_complex_table
```

---

load_mro	<i>load_mro</i>
----------	-----------------

---

**Description**

load\_mro

**Usage**

load\_mro()

**Value**

MRO in ontology\_index format

---

mro.obo	<i>MRO</i>
---------	------------

---

**Description**

mro.obo: MHC Restriction Ontology obo file.

**Usage**

mro.obo

**Format**

An object of class ontology\_index of length 18.

---

parse_allele_freq_html	<i>parse_allele_freq_html</i>
------------------------	-------------------------------

---

**Description**

format the allele frequency table and select columns of interest

**Usage**

parse\_allele\_freq\_html(allele\_freq\_table)

**Arguments**

allele\_freq\_table  
allele frequency table parsed from AFND website

**Value**

allele\_freq\_table reformatted

parse\_haplotype\_freq\_html  
*parse\_haplotype\_freq\_html*

**Description**

format the haplotype frequency table and select columns of interest

**Usage**

parse\_haplotype\_freq\_html(haplotype\_freq\_table)

**Arguments**

haplotype\_freq\_table  
haplotype frequency table parsed from AFND website

**Value**

haplotype\_freq\_table reformatted

plot\_allele\_frequency *Plotting allele frequencies*

**Description**

plot\_allele\_frequency Generate a World map displaying the frequency of a given table of HLA alleles. Use the function [query\\_allele\\_frequencies](#) to generate a table with allele frequencies.

**Usage**

plot\_allele\_frequency(allele\_frequency)

**Arguments**

allele\_frequency  
returned by [query\\_allele\\_frequencies](#)

**Value**

ggplot2 object displaying the allele frequencies on a world map.

**Examples**

```
# select frequency of given allele
sel_allele_freq <- query_allele_frequencies(hla_selection = "A*02:01",
hla_sample_size_pattern = "bigger_than",
hla_sample_size = 10000, standard="g")

plot_allele_frequency(sel_allele_freq)
```

---

```
query_allele_frequencies
      Query allele frequencies
```

---

**Description**

Query allele frequencies

**Usage**

```
query_allele_frequencies(
  hla_locus = NA,
  hla_selection = NA,
  hla_population = NA,
  hla_country = NA,
  hla_region = NA,
  hla_ethnic = NA,
  hla_sample_size_pattern = NA,
  hla_sample_size = NA,
  standard = "a"
)
```

**Arguments**

hla_locus	HLA locus that will be used for filtering data. A, B, C, DPA1, DPB1, DQA1, DQB1, DRB1
hla_selection	Allele that will be used for filtering data. e.g. A*01:01
hla_population	Numeric identifier of the population that will be used for filtering. This identifier is defined by the Allele Frequency Net Database.
hla_country	Country of interest (e.g. Germany, France, ...).
hla_region	Geographic region of interest (e.g. Europe, North Africa, ...)
hla_ethnic	Ethnic origin of interest (e.g. Caucasoid, Siberian, ...)

hla_sample_size_pattern	Keyword used to define the filtering for a specific population size. e.g. "bigger_than", "equal", "less_than", "less_equal_than", "bigger_equal_than"
hla_sample_size	Integer number used to define the filtering for a specific population size, together with the hla_sample_size_pattern argument.
standard	Population standards, as defined in the package vignette. "g" - gold, "s" - silver, "a" - all

**Value**

data.frame object containing the result of the allele frequency query

**Examples**

```
# select frequencies of the A*02:01 allele,
# for gold standard population with more than 10,000 individuals
sel <- query_allele_frequencies(hla_selection = "A*02:01",
hla_sample_size_pattern = "bigger_than", hla_sample_size = 10000,
standard="g")
```

---

query\_haplotype\_frequencies

*Query haplotype frequencies*

---

**Description**

Query haplotype frequencies

**Usage**

```
query_haplotype_frequencies(
  hla_selection = NA,
  hla_population = NA,
  hla_country = NA,
  hla_region = NA,
  hla_ethnic = NA,
  hla_sample_size_pattern = NA,
  hla_sample_size = NA
)
```

**Arguments**

**hla\_selection** Alleles that will be used to build the haplotype query. One entry per locus. If no entry for a given locus, the function will search for haplotypes that do not include specifications for this locus. If any allele for a given locus should be considered, the list entry should be "A\*" or other locus in same format.

hla_population	Numeric identifier of the population that will be used for filtering. This identifier is defined by the Allele Frequency Net Database.
hla_country	Country of interest (e.g. Germany, France, ...).
hla_region	Geographic region of interest (e.g. Europe, North Africa, ...)
hla_ethnic	Ethnic origin of interest (e.g. Caucasoid, Siberian, ...)
hla_sample_size_pattern	Keyword used to define the filtering for a specific population size. e.g. "bigger_than", "equal", "less_than", "less_equal_than", "bigger_equal_than"
hla_sample_size	Integer number used to define the filtering for a specific population size, together with the hla_sample_size_pattern argument.

**Value**

data.frame object containing the result of the allele frequency query

**Examples**

```
# works only for one haplotype at a time
query_haplotype_frequencies(hla_selection = c("A*02:01", "B*", "C*"),
hla_region = "Europe")
```

---

query\_population\_detail

*Query population metainformation*

---

**Description**

Query population metainformation

**Usage**

```
query_population_detail(population_ids)
```

**Arguments**

population\_ids List of numeric identifiers of the population that will be used for filtering. The identifier is defined by the Allele Frequency Net Database.

**Value**

data.frame object containing the result of the population detail query

**Examples**

```
population_detail <- query_population_detail(0001986)
```

read\_complete\_freq\_table  
*read\_complete\_freq\_table*

---

**Description**

read\_complete\_freq\_table

**Usage**

read\_complete\_freq\_table(url, type)

**Arguments**

url                   URL of the website containing frequency table  
type                  ["allele"|"haplotype"]

**Value**

frequency table

---

read\_population\_detail  
*read\_population\_detail*

---

**Description**

read\_population\_detail

**Usage**

read\_population\_detail(url, population\_id)

**Arguments**

url                   url of the page containing population information  
population\_id        numeric population identifier

**Value**

population information

---

retrieve\_chain\_lookup\_table  
*Retrieve MHC chain lookup table*

---

**Description**

Retrieve MHC chain lookup table

**Usage**

```
retrieve_chain_lookup_table(organism)
```

**Arguments**

organism            name of organism (e.g. "human")

**Value**

Table containing MHC chain information for the organism. It contains chain names, MHC restriction and protein sequence.

**Examples**

```
retrieve_chain_lookup_table("mouse")
```

---

verify\_parameters        *verify\_parameters*

---

**Description**

verify\_parameters

**Usage**

```
verify_parameters(  
  hla_locus,  
  hla_selection,  
  hla_population,  
  hla_country,  
  hla_region,  
  hla_ethnic,  
  hla_sample_size_pattern,  
  hla_sample_size,  
  standard = "a",  
  query_type  
)
```

**Arguments**

hla_locus	locus of hla frequencies to query
hla_selection	HLA alleles used for selection
hla_population	population id
hla_country	country used for allele frequency selection
hla_region	geographical region used for allele frequency selection
hla_ethnic	ethnic origin used for allele frequency selection
hla_sample_size_pattern	one of "bigger_than", "equal", "less_than", "less_equal_than", "bigger_equal_than", "different"
hla_sample_size	integer number used for population size
standard	one of "a" - all, "s" - silver, "g" - gold
query_type	"allele" or "haplotype"

**Value**

boolean to indicate, whether tests were passed

# Index

- \* **datasets**
  - human\_protein\_complex\_table, 16
- \* **internal**
  - check\_geographics, 4
  - check\_hla\_locus, 5
  - check\_hla\_selection, 5
  - check\_population, 6
  - check\_sample\_size, 6
  - check\_standard, 7
  - create\_encode\_handle, 7
  - extract\_population\_id, 9
  - extract\_population\_info, 9
  - extract\_population\_name, 10
  - extract\_sample\_info, 10
  - fetch\_encoded\_MAC, 11
  - get\_external\_file, 12
  - get\_nb\_pages, 14
  - get\_valid\_geographics, 15
  - getURL, 11
  - load\_mro, 17
  - mro.obo, 17
  - parse\_allele\_freq\_html, 17
  - parse\_haplotype\_freq\_html, 18
  - read\_complete\_freq\_table, 22
  - read\_population\_detail, 22
  - verify\_parameters, 23
- assemble\_protein\_complex, 3
- build\_allele\_group, 4
- check\_geographics, 4
- check\_hla\_locus, 5
- check\_hla\_selection, 5
- check\_population, 6
- check\_sample\_size, 6
- check\_standard, 7
- create\_encode\_handle, 7
- decode\_MAC, 8
- encode\_MAC, 8
- extract\_population\_id, 9
- extract\_population\_info, 9
- extract\_population\_name, 10
- extract\_sample\_info, 10
- fetch\_encoded\_MAC, 11
- get\_external\_file, 12
- get\_G\_group, 12
- get\_mhspan\_input, 13
- get\_nb\_pages, 14
- get\_P\_group, 14
- get\_serotypes, 15
- get\_valid\_geographics, 15
- get\_valid\_organisms, 16
- getURL, 11
- human\_protein\_complex\_table, 16
- load\_mro, 17
- mro.obo, 17
- parse\_allele\_freq\_html, 17
- parse\_haplotype\_freq\_html, 18
- plot\_allele\_frequency, 18
- query\_allele\_frequencies, 18, 19
- query\_haplotype\_frequencies, 20
- query\_population\_detail, 21
- read\_complete\_freq\_table, 22
- read\_population\_detail, 22
- retrieve\_chain\_lookup\_table, 23
- verify\_parameters, 23