

Package: Rduckvep (via r-universe)

October 21, 2026

Title DuckDB Variant Consequence Annotation

Version 0.1.0

Description Builds and loads the bundled DuckVEP C extension for relation-first variant consequence and phased-haplotype annotation.

License GPL (>= 2)

URL <https://rgenicsetl.github.io/DuckVEP/Rduckvep/>,
<https://github.com/RGenomicsETL/DuckVEP>

BugReports <https://github.com/RGenomicsETL/DuckVEP/issues>

Copyright See inst/COPYRIGHT

Encoding UTF-8

Depends R (>= 4.1)

Imports DBI, duckdb

Suggests tinytest, Rduckhts

SystemRequirements cmake, GNU make, zlib

Config/Needs/website pkgdown

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make xz-utils zlib1g-dev

Repository <https://rgenicsetl.r-universe.dev>

Date/Publication 2026-10-21 14:11:47 UTC

RemoteUrl <https://github.com/RGenomicsETL/DuckVEP>

RemoteRef HEAD

RemoteSha 45a5b94d2b189695f459a13719bcb99587ca3a00

RemoteSubdir r/Rduckvep

Contents

rduckvep_annotate	2
rduckvep_builders	3
rduckvep_coding_calls	5
rduckvep_coding_transcripts	6
rduckvep_connect	7
rduckvep_haplotypes	8
rduckvep_load	13
rduckvep_load_model	13
rduckvep_prepare_breakend_fusion	14
rduckvep_prepare_breakend_pairs	15
rduckvep_prepare_expansionhunter	15
rduckvep_prepare_structural_hgvs	16
rduckvep_prepare_sv_geometry	17
rduckvep_product_scenario_v1	17
rduckvep_save_model	18
Index	20

rduckvep_annotate	<i>Annotate events using a loaded DuckVEP model</i>
-------------------	---

Description

Annotate events using a loaded DuckVEP model

Usage

rduckvep_annotate(con, events_table, model_name, ...)

Arguments

con	A DBI connection with DuckVEP loaded.
events_table	Event relation name.
model_name	Loaded model name.
...	Named builder options (for annotation: 'hgvs', 'rich', 'upstream_distance', 'downstream_distance'; for Ensembl regions and transcripts: 'species_id'; for receipts: 'regulation_features_table'; for the loss-of-function relation: 'gerp', 'ancestor', 'phylocaf', 'min_intron_size', 'gerp_end_trunc_cutoff', 'check_complete_cds'). Other builders accept no options. Each value must be a length-one string, logical, integer, double or 'NA'.

Value

A data frame of annotated events.

rduckvep_builders	<i>Generate DuckVEP SQL on the caller's connection</i>
-------------------	--

Description

These functions invoke the extension's native SQL builders. The returned SQL can be passed to 'DBI::dbGetQuery()' or used in a 'CREATE TABLE AS' statement. Required relation names are passed as strings; the native builders quote them. Optional named arguments are forwarded as a DuckDB STRUCT. Load the extension on 'con' with [rduckvep_load()] before calling a builder.

Usage

```
rduckvep_annotate_sql(con, events_table, model_name, ...)
```

```
rduckvep_transcript_projection_sql(  
  con,  
  events_table,  
  annotations_table,  
  transcripts_table,  
  ...  
)
```

```
rduckvep_lof_sql(  
  con,  
  annotations_table,  
  transcripts_table,  
  reference_table,  
  ...  
)
```

```
rduckvep_ensembl_regions_sql(  
  con,  
  core_schema,  
  reference_chunks_table,  
  assembly,  
  ...  
)
```

```
rduckvep_ensembl_transcripts_sql(  
  con,  
  core_schema,  
  reference_chunks_table,  
  assembly,  
  ...  
)
```

```
rduckvep_ensembl_regulation_features_sql(  
  con,  
  core_schema,  
  reference_chunks_table,  
  assembly,  
  ...  
)
```

```

    con,
    funcgen_schema,
    regions_table,
    ...
)

rduckvep_model_receipt_sql(
    con,
    regions_table,
    transcripts_table,
    source_name,
    source_version,
    assembly,
    source_manifest_sha256,
    reference_sha256,
    transcript_filter,
    ...
)

```

Arguments

<code>con</code>	A DBI connection with DuckVEP loaded.
<code>events_table</code>	Event relation name.
<code>model_name</code>	Loaded model name.
<code>...</code>	Named builder options (for annotation: 'hgvs', 'rich', 'upstream_distance', 'downstream_distance'; for Ensembl regions and transcripts: 'species_id'; for receipts: 'regulation_features_table'; for the loss-of-function relation: 'gerp', 'ancestor', 'phyloCSF', 'min_intron_size', 'gerp_end_trunc_cutoff', 'check_complete_cds'). Other builders accept no options. Each value must be a length-one string, logical, integer, double or 'NA'.
<code>annotations_table</code>	Annotation relation name.
<code>transcripts_table</code>	Transcript relation name.
<code>reference_table</code>	Reference sequence chunk relation ('chrom', 'start', 'end', 'seq').
<code>core_schema</code>	Ensembl core schema name.
<code>reference_chunks_table</code>	Reference chunk relation name.
<code>assembly</code>	Assembly name.
<code>funcgen_schema</code>	Ensembl funcgen schema name.
<code>regions_table</code>	Canonical region relation name.
<code>source_name, source_version</code>	Source identity strings.
<code>source_manifest_sha256, reference_sha256</code>	Source and reference digests.

transcript_filter
Transcript selection description.

Details

Model receipts include nonempty original RNA coding authority in their content hash. Absent or all-NULL authority preserves legacy receipt identity. A receipt commits to supplied content, not independently verified biological correctness.

Value

A length-one SQL string.

rduckvep_coding_calls *Read the Coding Calls of a VCF or BCF File*

Description

The fused native route into [rduckvep_haplotypes()]: one pass over a (bgzipped) VCF or BCF file with the bundled HTSlib. Each record is mapped to the model's 'seq_region' by 'seq_region_name', each ALT allele goes through the same discovery as [rduckvep_coding_transcripts()], and only a record that touches coding sequence has its FORMAT parsed and its genotypes decoded. The other records, about 99% of a whole genome, cost one line read and a few interval lookups.

Usage

```
rduckvep_coding_calls(
  con,
  model_name,
  path,
  table_name = NULL,
  overwrite = FALSE
)
```

Arguments

con	An open DuckDB connection with DuckVEP loaded.
model_name	Name of an already loaded DuckVEP model (loaded with 'seq_region_name' in its regions query) without wrapped circular objects.
path	Path of the VCF, bgzipped VCF or BCF file.
table_name	Optional output table name.
overwrite	Whether to replace an existing output table.

Details

The result is the calls relation that [rduckvep_haplotypes()] consumes: one row per ALT allele, transcript and sample with 'event_index' (record ordinal times 64 plus the ALT ordinal minus one, where the ordinal counts every data record of the file), 'seq_region', 'position', 'reference', 'alternate', 'alt_index', 'transcript_index', 'sample_index' (the sample column, from 0), 'alleles' (missing alleles are NA), 'phase_before' (a leading 'FALSE', then one flag per separator, 'I' being 'TRUE') and 'phase_set' (the integer 'PS' of the sample; labels such as 'PATMAT' and a missing 'PS' give NA). Records on contigs the model does not know, symbolic ALTs and records with no ALT make no rows. A file without samples, without a 'FORMAT/GT' header line, a coding record without 'GT', more than 64 ALT alleles in a coding record, an invalid 'POS' and a model loaded without 'seq_region_name' are errors.

Value

A data frame of the calls, or invisible 'TRUE' when creating 'table_name'. To feed [rduckvep_haplotypes()], create the table and pass "SELECT * FROM <table_name>".

```
rduckvep_coding_transcripts
```

Find the Transcripts Whose Coding Sequence Events Overlap

Description

The fast discovery route into [rduckvep_haplotypes()]: one native lookup in the resident transcript interval index and exon layout instead of annotating every record. Each row of 'events_query' is one VCF record and one ALT allele with its shared anchor base ('reference', 'alternate', one-based 'position'); the event is normalized exactly as the annotation builder normalizes it, so the pairs returned are exactly those to which [rduckvep_annotate()] assigns the CDS region bit: an indel's anchor base alone never creates an overlap, an insertion is placed on the base the annotation builder places it on (and is not a CDS event at the outer CDS edges), and an intron of at most 13 bases inside the CDS counts as coding. Symbolic and breakend ALTs, missing ALTs and alleles identical to REF are not events. Records with no coding overlap, the large majority of a genome, produce no rows, so a VCF reduces to its coding records before any per-record work. The result is the 'transcript_index' column of a calls relation for [rduckvep_haplotypes()]: join it with the records' genotypes.

Usage

```
rduckvep_coding_transcripts(
  con,
  events_query,
  model_name,
  table_name = NULL,
  overwrite = FALSE
)
```

Arguments

con	An open DuckDB connection with DuckVEP loaded.
events_query	One nonempty SELECT query supplying 'event_index', 'seq_region', 'position', 'reference' and 'alternate' (any integer types for the first three).
model_name	Name of an already loaded DuckVEP model without wrapped circular objects.
table_name	Optional output table name.
overwrite	Whether to replace an existing output table.

Value

A data frame of 'event_index' and 'transcript_index' (one row per pair, ascending by transcript within an event), or invisible 'TRUE' when creating 'table_name'.

rduckvep_connect	<i>Open DuckDB with DuckVEP loaded</i>
------------------	--

Description

Open DuckDB with DuckVEP loaded

Usage

```
rduckvep_connect(dbdir = ":memory:", extension_path = NULL)
```

Arguments

dbdir	Database path, or ':memory:'.
extension_path	Optional locally built extension artifact.

Value

A DBI connection. Disconnect with 'shutdown = TRUE'.

rduckvep_haplotypes *Replay Phased Transcript Haplotypes*

Description

Consume a SELECT query of flat event-by-transcript-by-sample calls through the bundled native replay stream. DuckDB derives complete phase-set domains and sorts the calls. Each output row is one occupied shared path, with CDS, protein, carrier keys and all contributing events. Incomplete calls and projection/edit failures retain provenance without inventing sequence.

Usage

```
rduckvep_haplotypes(
  con,
  calls_query,
  model_name,
  phase_policy = c("strict", "vep_compat"),
  ...,
  input_mode = c("alt_events", "source_records"),
  hgvs = FALSE,
  table_name = NULL,
  overwrite = FALSE
)
```

Arguments

con	An open DuckDB connection with DuckVEP loaded.
calls_query	One nonempty SELECT query supplying the call relation.
model_name	Name of an already loaded DuckVEP model.
phase_policy	Strict GT/PS interpretation, or ‘vep_compat’ for the called-slot order of the pinned executable VEP release. Decoded missing calls remain incomplete; source-record input uses the pinned raw parser and explicitly conditional missing-slot interpretation.
...	Named positive integer workspace capacities accepted by ‘duckvep_haplotypes’, such as ‘max_active_events’, ‘max_active_carriers’, ‘max_sequence_bases’, ‘max_ploidy’, ‘max_phase_sets’, and ‘workspace_limit’.
input_mode	‘alt_events’ for decoded per-ALT calls, or ‘source_records’ for raw GT and complete source ALT lists under ‘vep_compat’.
hgvs	Whether to request bounded protein HGVS for supported completed paths.
table_name	Optional output table name.
overwrite	Whether to replace an existing output table.

Details

`'nominal_length_diff'` is the signed sum of projected replacement ALT-minus-REF lengths before clipping at the current CDS end. It is zero for reference-only replay and NA without a CDS. Known and conditional paths retain this value; overlapping replacements can produce a different rebuilt CDS length change.

`'coding_blocks'` groups physical edits that share an alternate codon or displace then restore the reading frame. Each block gives one-based reference `'cds_start'`, transcript-oriented `'reference'`/`'alternate'` spans, zero-based `'alt_start0'` in the rebuilt CDS, `'length_change'`, `'sequence_flags'` and `'event_indices'`: one source event ID per physical edit in ascending CDS order. An ID can repeat for several differing islands, within or across blocks; the list length is the physical edit count. Join IDs to `'contributors'` for raw alleles and input provenance. Spans include retained bases between edits; they are not aligned differences or HGVS normalization. An insertion has empty reference; a deletion has empty alternate. Unknown sequences have NULL blocks, not an empty known result. Blocks reuse `'max_leaf_edits'` capacity and the per-call workspace limit.

`'cds_differences'` instead contains aligned differing runs: zero-based `'ref_start0'`, `'alt_start0'`, and `'alignment_start0'`, with borrowed sequence spans materialized as `'reference'` and `'alternate'`. An empty span denotes a gap. The reference uses replay's uppercase DNA spelling; model bytes stay immutable. Runs join only when adjacent columns have the same gap/non-gap type on both sides. Indel-bearing paths use the VEP-116 pure-Perl global-alignment score and tie order; substitution-only paths compare corresponding positions. Repeated sequence can therefore place a difference away from its contributing event. Differences are not HGVS normalization or event-provenance reassignment. Unknown sequences have NULL differences. `'max_alignment_cells'` bounds the exact traceback band and `'max_leaf_differences'` bounds output runs; exhaustion is an error, not approximate alignment or discarded differences.

`'protein_differences'` has the same span fields and alignment mode, with positions in amino acids. Its reference follows Ensembl-116 start-methionine, terminal-stop and curated single-residue peptide-edit rules. Haplosaurus then appends `'*'` only for an exact uppercase TAA/TAG/TGA raw-CDS suffix, including nonstandard-table and partial-CDS cases. The alternate is the displayed first-stop prefix without reference peptide edits. Unknown paths and reference CDS shorter than one codon have NULL protein differences; identical known proteins have an empty list. Both difference axes reuse the same native scratch and are separately subject to the alignment-cell and run limits.

`'hgvs = TRUE'` requests a VEP-116-derived protein HGVS suffix in `'hgvsp'`: equality, one operation, or a cis allele such as `'p.[(Gly2del;Ala4CysfsTer2)]'`. Protein operations use the completed path and may combine several physical edits; source contributors and coding blocks remain unchanged. `'hgvsp_status'` is `'ok'` only after the complete operation set has been rendered. Other statuses retain NULL text without discarding the sequence or provenance. Conditional or unavailable sequence, ordered overlapping replacements, missing peptide data, unsupported coding contexts, and unrepresentable protein ends remain explicit. Phase policy controls allele assignment, not HGVS nomenclature. Protein HGVS retains VEP's local-peptide and unknown-residue presentation. An `'ok'` status reports a supported computation, not independent HGVS-rule certification. A path with one original ALT source uses independent-event VEP-116 HGVS, including genomic shifting and absent results. Multiple differing islands within one MNV retain that single source identity. Placement needing an unavailable genomic FASTA returns `'missing_reference'`. Prepared references retain their own residues and length without changing raw CDS/frame facts or inventing source edits. Loss of only a reference stop marker supplies no alternate extension, and insertions require reference flanks. `'max_hgvs_operations'` bounds the working operation stack and final operations; `'max_hgvs_bytes'` bounds text bytes excluding NUL;

'max_hgvs_reference_bytes' bounds the query-local FASTA result buffer, including NUL and line-ending scratch. Sequence/edit scratch derives from 'max_sequence_bases' and 'max_leaf_edits', and allele scratch from 'max_allele_bytes' and the literal allele width. All DuckVEP-owned buffers count toward 'workspace_limit'; HTSLib handle/transport storage is separate. Exhaustion errors instead of truncating text. Disabled HGVS allocates no HGVS buffers or reference handle and returns 'not_requested'. Identifiers are joined through the model transcript ordinal; the suffix contains no accession.

'prediction_policy' names the versioned contract ('duckvep-coding'). 'prediction_status' and 'prediction_reason' state eligibility for its supported domain and whether the classifiers decided it (see below), and 'nmd_prediction' gives the rule 'ejc50' NMD prediction; 'nmd_exceptions' lists the exceptions to that rule that apply to a premature stop ('start_proximal', 'long_exon') without changing the prediction. On a decoded eligible path, a lost stop extends 'protein' through the stored 3' flank to the next stop. Raw source replay retains its compatibility protein without that extension. The ordinary coding domain is strict complete phased calls of any ploidy (a haploid call has one lane), one unambiguous heterozygous phase domain per transcript and sample (an unphased heterozygous call that is the sample's only heterozygous call on the transcript needs no phase), literal ACGT SNVs, MNVs and indels of any length, non-overlapping and inside coding exons of CDSs in a supported genetic code with no curated RNA or peptide edits. A complete CDS needs a start codon (ATG in the standard code) and a terminal stop; a CDS whose start or end is not annotated is predicted without the test for the missing end. Statuses: 'incomplete_input' (reasons 'missing_call', 'unphased_heterozygous', 'unresolved_cross_ps_phase'), 'edit_conflict' ('contradictory_edits'), 'unsupported_overlap' ('overlapping_edits', 'duplicate_edits', 'ambiguous_same_gap_insertions') and 'unsupported_context' ('non_strict_phase_policy', 'non_literal_allele', the transcript reasons 'transcript_not_coding', 'non_standard_codon_table', 'curated_transcript', 'incomplete_cds', 'noncanonical_start', 'noncanonical_stop', 'internal_stop', and preserved projection or sequence reasons such as 'reference_mismatch', 'outside_cds', 'non_contiguous', 'invalid_allele'). Phase-domain completeness and ploidy belong to each sample/phase/lane, so 'carrier_predictions' gives the keyed result in 'carriers' order; the row status is 'predicted' or 'eligible_classifier_pending' only when every carrier is eligible, and otherwise the first ineligible carrier's result.

A bounded literal overlap profile composes agreeing substitutions, identical indels, identical minimal delins and compatibly placed pure deletions in one coding exon shared by every contributing source. Disjoint sources in other exons remain outside this overlap profile. Opaque delins aliases apply one replacement without assigning ancestry to ALT bases; every original REF, including padding, must match. Shared edits count once while each source retains its identity. Ambiguous placements and distinct interacting delins remain unsupported. Immediate introduced stops in compound protein HGVS are nonsense substitutions, not multi-residue delins. Separate complete-RNA and typed-curation profiles are described in the function reference and haplotype contract.

Raw 'source_records' under 'vep_compat' can qualify for coding prediction when every original call is complete diploid, with at most one original REF/ALT heterozygote and homozygous background. Each retained slot must map to its original copy; those copies may follow different disjoint coding paths. Multiallelic 0/2 is one original heterozygote; 1/2 is outside this profile. CALLED alone, hypothetical slots, leading separators, non-diploid or missing calls do not prove copies. Omitted non-diploid REF calls also prevent proof. Reference-only compatibility rows retain their status rather than becoming ALT predictions. Complete and unproven carriers occupy separate qualification rows, preserving the complete sample's outer prediction even when other samples share its sequence. Row grouping can change even when another context prevents prediction. Qualification-only carriers consume bounded carrier slots; per-carrier raw CDS/protein/HGVS stay unchanged.

'haplotype_consequences' (a character list) and 'haplotype_impact' give the reduced whole-protein SO set and IMPACT of the edited coding sequence, decided by the same-codon and frame/stop-gain classifiers for an eligible path; both are NULL unless 'prediction_status' is 'predicted'. The combined haplotype is classified from its translated sequence, never one edit alone and never from the nominal net length change. A new first stop upstream of the reference terminator is 'stop_gained' (HIGH), and 'frameshift_variant' (HIGH) is added when that stop codon intersects a displaced-frame interval; with no stop at all, a frame still displaced when the CDS runs out is 'frameshift_variant' and 'stop_lost'. A frame that is displaced and restored before termination has no frame term: an unchanged peptide is 'synonymous_variant' (LOW), any other change 'protein_altering_variant'. Two compensating indels can therefore be rescued or end in an earlier stop. Without frame edits, substitutions that change the peptide are 'missense_variant', and edits that are all pure insertions or all pure deletions are 'inframe_insertion' or 'inframe_deletion'; any other frame-preserving replacement is 'protein_altering_variant' (MODERATE). Edits after the first stop stay contributors, never expressed effects. Cis and trans edits are separate carriers of separate paths. 'carrier_predictions' also carries 'haplotype_impact' and 'haplotype_consequences' per carrier (the edited sequence is shared, so its set is defined for every predicted carrier), so a row with an ineligible carrier (for example a triploid call) has NULL row-level fields but keeps the set of its eligible carriers. The start/stop classifier decides the remaining paths, in this order. An edited CDS that no longer begins with ATG is 'start_lost' (HIGH) alone: initiation and NMD are unknown, so start loss suppresses every other prediction. An edited CDS with no stop is 'stop_lost' (HIGH), plus 'frameshift_variant' when the frame is still displaced when the CDS runs out; no downstream extension is invented. A first stop before the homologous reference terminator that truncates the peptide is 'stop_gained', as above. A first stop at the terminator with an unchanged peptide is 'stop_retained_variant' (LOW) when the terminal codon changed or moved (it precedes 'synonymous_variant'), and any other change keeps the whole-protein category above; a stop inserted next to the terminator is judged the same way. Every eligible path is therefore decided and the reason 'start_stop_classifier_pending' no longer occurs. Reference lanes have no carrier row. 'contributor_provenance' lists every source of the path, including omitted, shadowed, unapplied (conflicting or incomplete) and post-stop ones, with its original operands, 'alt_index', evidence, projection status, 'role' and differing edit count. The role is assigned per edit by its position relative to the first stop of the edited sequence: a source is 'post_stop' only when every edit it contributes starts after that stop, even if it shares an interaction block with an earlier edit. 'normalized_edits' lists the differing CDS edit islands in ascending order with their source 'event_index' and 'coding_blocks' index (NULL when no sequence was rebuilt). Sequence deduplication never merges or drops contributors. Malformed identities and exhausted budgets error.

'nmd_prediction' is the versioned NMD prediction of the edited transcript under rule 'nmd_rule' ('ejc50'), an EJC-distance heuristic on the whole haplotype, never per allele, never a union of single-allele results and not the VEP NMD plugin. For a newly premature stop ('stop_gained' in 'haplotype_consequences', with or without 'frameshift_variant') it is 'trigger' when $J - S > 50$ and 'escape' otherwise. S ('nmd_stop_position') is the final nucleotide of the first stop codon and J ('nmd_junction_position') the final nucleotide of the penultimate exon, both 1-based in the edited spliced-transcript (cDNA, 5' UTR included) coordinates, so an indel before the stop moves S, an indel before or inside the penultimate exon moves J, and one in the last exon moves neither; an edit after the stop still moves J when it lies at or before the penultimate exon's last base, while S never moves. A single-exon transcript is 'escape' and has no 'nmd_junction_position'. 'not_applicable' is known termination without a new premature stop (synonymous, missense, protein-altering, in-frame, 'stop_retained_variant', a frame displaced and restored before termination) and a reference lane, which has no termination change (strict decoded input gives an ALT-free lane no carrier, so no

such row is returned). ‘unknown’ is lost initiation (‘start_lost’), an edited CDS with no stop (a frame that runs off the CDS and ‘stop_lost’: the termination is unavailable, not known absent), unresolved exon topology, and every path that is not ‘predicted’ (‘incomplete_input’, ‘edit_conflict’, ‘unsupported_overlap’, ‘unsupported_context’, and the non-strict phase policy), whose consequences and IMPACT are NULL as before. ‘nmd_stop_position’ and ‘nmd_junction_position’ are NULL unless a premature stop was decided (J also needs a junction). ‘nmd_contributors’ lists the ‘event_index’ of the contributors that jointly determine a ‘trigger’ or ‘escape’: those with role ‘applied’ (the edits up to and including the stop) plus the ‘post_stop’ indels that changed length at or before the penultimate exon’s last base, i.e. exactly the edits that moved J. Their ‘role’ stays ‘post_stop’. Post-stop substitutions, post-stop indels in the last exon, and ‘shadowed’ or ‘omitted’ sources are not listed. For ‘not_applicable’, ‘unknown’ and every non-predicted row it is NULL, as are the stop and junction positions. The row fields are ‘unknown’ and NULL unless every carrier is predicted; the keyed ‘carrier_predictions’ also carry ‘nmd_prediction’, ‘nmd_stop_position’ and ‘nmd_junction_position’, so one ineligible carrier does not hide the others. Only the EJC distance is modelled: no reinitiation, no long-exon exception and no ‘NMD_transcript_variant’ biotype term.

‘stop_in_displaced_frame’ reports whether any of the first translated stop codon’s three bases overlaps a frame-displaced span of the rebuilt CDS. Displacement starts at a frame-changing edit and ends after the alternate bases of the restoring edit, or continues downstream if unrestored. It is FALSE for no stop or a stop after frame restoration, and NA when sequence is unavailable. It is a sequence fact, not a combined SO consequence or a claim that a restored DNA frame rescues the protein.

Whole-haplotype DNA HGVS, complete protein HGVS and structural-event composition remain unfinished. Input must contain one row per ‘event_index’, ‘transcript_index’, ‘sample_index’, with columns ‘seq_region’, ‘position’, ‘reference’, ‘alternate’, ‘alt_index’, ‘alleles’, ‘phase_before’ and nullable ‘phase_set’. Event indices identify individual ALT events; retain their source-record mapping. Transcript ordinals belong to the named model. Candidate selection is explicit in this input relation.

With ‘input_mode = "source_records"’, the required columns are ‘event_index’, ‘seq_region’, ‘position’, ‘reference’, ‘alternates’ (a character list), ‘transcript_index’, ‘sample_index’, and ‘gt’ (original VCF text). Here ‘event_index’ identifies the whole source record. This mode requires ‘vep_compat’: the pinned file profile consumes two slots and ignores PS. Missing calls and undefined slots can yield ‘conditional’ sequence with evidence bit 8; this is not known phase or biological rescue. Contributor ‘alt_index’ is 0 for REF, a positive source ALT ordinal, or NA for an undefined slot’s full-REF deletion. Source ALT strings must be nonempty and nonmissing. Projection failures still withhold sequence. Raw spelling must be retained at ingestion; it cannot be reconstructed losslessly from decoded GT arrays.

Preparation reads committed objects on the registry’s retained connection; caller-local temporary objects and uncommitted changes are not visible. One preparation may run per registry at a time. Nested or concurrent preparation returns a busy error; completed scans use independent native workspaces.

Value

A data frame, or invisible ‘TRUE’ when creating ‘table_name’.

rduckvep_load	<i>Load the bundled DuckVEP extension</i>
---------------	---

Description

Load the bundled DuckVEP extension

Usage

```
rduckvep_load(con, extension_path = NULL)
```

Arguments

con An open DuckDB connection with unsigned extension loading enabled.
extension_path Optional extension artifact for this DuckDB version.

Value

Invisibly, the connection.

rduckvep_load_model	<i>Load a resident transcript model, on either DuckDB C API host</i>
---------------------	--

Description

The v1 host runs the three (plus optional) relation queries on a private connection with ‘duckvep_model_load()’. The v2 host cannot: its model sink is a COPY format, so the same load is a sequence of ‘COPY’ statements into staging followed by ‘duckvep_model_publish()’, and those statements come from ‘duckvep_model_load_sql()’. This function picks the host by what the loaded extension provides and hides the difference; the result is the same resident model. One visible difference remains: on v2 the queries run on ‘con’ in its own transaction, so temporary tables and uncommitted rows are visible to them; on v1 the queries run on a separate connection and see only committed, permanent relations.

Usage

```
rduckvep_load_model(
  con,
  name,
  regions_query,
  transcripts_query,
  exons_query,
  mature_mirna_query = NULL,
  peptide_edit_query = NULL,
  interval_feature_query = NULL,
```

```

reference_fasta = NULL,
transcript_coverage_complete = NULL,
rna_edit_query = NULL
)

```

Arguments

con	A DBI connection with DuckVEP loaded.
name	Model name (non-empty, not yet loaded).
regions_query, transcripts_query, exons_query	SELECT statements that return the typed, ordered relations (see the function reference).
mature_mirna_query, peptide_edit_query, interval_feature_query	Optional SELECT statements for the optional relations.
reference_fasta	Optional path of an indexed reference FASTA.
transcript_coverage_complete	Optional logical ('NA' is an error).
rna_edit_query	Optional SELECT statement for typed transcript RNA edits.

Value

'TRUE', invisibly, when the model is loaded.

rduckvep_prepare_breakend_fusion

Combine BND identity with endpoint genes

Description

'pairs_table' holds the result of [rduckvep_prepare_breakend_pairs()] and 'genes_table' has 'event_index' and 'gene_id', one row per gene overlapped by that physical endpoint. Each source record keeps its own row and gene list. 'candidate_partner_genes' records partner-gene evidence only: the result never asserts a fusion, frame or phase.

Usage

```
rduckvep_prepare_breakend_fusion(con, pairs_table, genes_table, ...)
```

Arguments

con	A connection with DuckVEP loaded.
pairs_table	Name of a relation with the identity builder columns.
genes_table	Name of an 'event_index', 'gene_id' relation.
...	Named options for the native builder.

Value

A data frame with one row per physical source record.

```
rduckvep_prepare_breakend_pairs
```

Validate BND mate and event identity

Description

The input relation has 'event_index', 'chrom', 'pos', 'id', 'ref', 'alt' and 'info'. The native builder returns one row per physical source record and never merges a record with its mate. 'status' is 'reciprocal', 'unproven', 'conflict', 'invalid' or 'not_applicable'; 'reason' is a stable identifier. ALT syntax alone never proves a fusion, phase or inserted-only sequence: 'fusion_status' is always 'not_asserted' and 'phase_status' 'not_evaluated'.

Usage

```
rduckvep_prepare_breakend_pairs(con, events_table, ...)
```

Arguments

con	A connection with DuckVEP loaded.
events_table	Name of the caller's raw VCF record relation.
...	Named options for the native builder.

Value

A data frame with one identity row per input record.

```
rduckvep_prepare_expansionhunter
```

Prepare ExpansionHunter v5 repeat alleles

Description

The input relation has 'event_index', 'info', 'format', 'sample', 'ref', 'alt', and one-based 'alt_index'. The reference relation has 'event_index' and the literal 'reference_sequence' from POS+1 through END. Each input event retains its source, preparation reason and exactness decision.

Usage

```
rduckvep_prepare_expansionhunter(con, input_table, reference_table, ...)
```

Arguments

con	A connection with DuckVEP loaded.
input_table	Name of the caller's input relation.
reference_table	Name of the literal reference relation.
...	Named options for the native builder.

Value

A data frame with typed preparation rows.

rduckvep_prepare_structural_hgvs	<i>Prepare structural HGVS descriptions</i>
----------------------------------	---

Description

The events relation has 'event_index', 'chrom', 'pos', 'ref', 'alt' and 'info'; the reference relation has 'event_index' and 'reference_sequence' for 'pos' through 'END'. Exact-span symbolic DEL, DUP, DUP:TANDEM and INV alleles receive an unshifted genomic description and an equivalent literal allele for transcript HGVS. Everything else is explicitly 'unavailable' or 'unsupported', with a stable reason. VEP 116 itself emits no HGVS for symbolic structural alleles or breakends.

Usage

```
rduckvep_prepare_structural_hgvs(con, events_table, reference_table, ...)
```

Arguments

con	A connection with DuckVEP loaded.
events_table	Name of the caller's raw VCF record relation.
reference_table	Name of the reference sequence relation.
...	Named options for the native builder ('max_span').

Value

A data frame with one row per input event.

rduckvep_prepare_sv_geometry

Prepare VCF structural geometry and insertion provenance

Description

The input relation has ‘event_index’, ‘pos’, ‘ref’, ‘alt’, and ‘info’. Symbolic alleles retain break-point confidence bounds separately from the nominal coordinates. Literal anchored insertions carry inserted bases; symbolic INFO/SEQ is source provenance, not a literal insertion.

Usage

```
rduckvep_prepare_sv_geometry(con, input_table, ...)
```

Arguments

con	A connection with DuckVEP loaded.
input_table	Name of the caller’s input relation.
...	Named options for the native builder.

Value

A data frame with one typed preparation row per input event.

rduckvep_product_scenario_v1

Evaluate a supplied RNA product programme

Description

‘reference’ has ‘start1’, ‘dna’, and ‘strand’. ‘exons’ and ‘blocks’ are data frames with ‘start1’ and ‘end1’; blocks are in transcription order. ‘sources’ is a data frame with ‘source_id’, ‘start1’, ‘reference’, and ‘alternate’. ‘programme’ has ‘id’, ‘evidence_id’, ‘assumption_id’, ‘expected_reference_rna’, ‘annotated_start1’, ‘alternative_start1’, ‘alternative_trigger_id’, and ‘genetic_code’. Supply zero and an empty trigger when there is no alternative initiation programme.

Usage

```
rduckvep_product_scenario_v1(
  conn,
  reference,
  exons,
  blocks,
  sources,
  programme
)
```

Arguments

conn A DBI connection with the DuckVEP extension loaded.
 reference, exons, blocks, sources, programme
 Declared scenario inputs.

Details

Genomic positions and source IDs are unsigned 64-bit coordinates. Supply decimal character values for integers outside R's exact double range. The R result represents these unsigned 64-bit values as decimal character data. The edit set is hypothetical, not evidence that calls were observed in cis. The supplied RNA is validated against the reference and mapped blocks. Peptides begin with methionine at the declared valid initiator and end at the first reached stop; the returned RNA retains the complete mapped sequence. 'conditional_nmd' remains unknown without shared NMD eligibility rules. This experimental versioned API supports disjoint, fixed-length substitutions; it does not infer an RNA product or scan for alternative initiation.

Value

A one-row data frame with typed source and effect LIST columns.

Examples

```
local({
  con <- rduckvep_connect()
  on.exit(DBI::dbDisconnect(con, shutdown = TRUE))
  reference <- list(start1 = 1L, dna = "ATGAAATAA", strand = 1L)
  exons <- data.frame(start1 = 1L, end1 = 9L)
  sources <- data.frame(source_id = 0L, start1 = 4L,
                        reference = "A", alternate = "G")
  programme <- list(id = "annotated", evidence_id = "supplied",
                    assumption_id = "joint_assignment", expected_reference_rna = reference$dna,
                    annotated_start1 = 1L, alternative_start1 = 0L,
                    alternative_trigger_id = "", genetic_code = 1L)
  answer <- rduckvep_product_scenario_v1(con, reference, exons, exons,
                                         sources, programme)
  answer[c("programme_id", "allele_evidence", "product_evidence", "conditional_nmd")]
})
```

rduckvep_save_model	<i>Save and restore a resident model as a snapshot file</i>
---------------------	---

Description

'rduckvep_save_model()' writes a loaded model's native arrays to one file with 'duckvep_model_save()'. 'rduckvep_restore_model()' loads a model from that file with 'duckvep_model_restore()': the file is mapped read-only instead of copied, so restoring is several times faster than loading the relations, and every R session that restores the same file shares one copy of it in memory. A snapshot is validated when it is restored and is tied to the DuckVEP build that wrote it; a build with a different model layout refuses it. It records the path of the model's reference FASTA, not its bytes.

Usage

```
rduckvep_save_model(con, name, path)
```

```
rduckvep_restore_model(con, name, path)
```

Arguments

<code>con</code>	A DBI connection with DuckVEP loaded.
<code>name</code>	Model name: a loaded model to save, or a new name to restore under.
<code>path</code>	Path of the snapshot file.

Value

'TRUE', invisibly.

Index

`rduckvep_annotate`, [2](#)
`rduckvep_annotate_sql`
 (`rduckvep_builders`), [3](#)
`rduckvep_builders`, [3](#)
`rduckvep_coding_calls`, [5](#)
`rduckvep_coding_transcripts`, [6](#)
`rduckvep_connect`, [7](#)
`rduckvep_ensembl_regions_sql`
 (`rduckvep_builders`), [3](#)
`rduckvep_ensembl_regulation_features_sql`
 (`rduckvep_builders`), [3](#)
`rduckvep_ensembl_transcripts_sql`
 (`rduckvep_builders`), [3](#)
`rduckvep_haplotypes`, [8](#)
`rduckvep_load`, [13](#)
`rduckvep_load_model`, [13](#)
`rduckvep_lof_sql` (`rduckvep_builders`), [3](#)
`rduckvep_model_receipt_sql`
 (`rduckvep_builders`), [3](#)
`rduckvep_prepare_breakend_fusion`, [14](#)
`rduckvep_prepare_breakend_pairs`, [15](#)
`rduckvep_prepare_expansionhunter`, [15](#)
`rduckvep_prepare_structural_hgvs`, [16](#)
`rduckvep_prepare_sv_geometry`, [17](#)
`rduckvep_product_scenario_v1`, [17](#)
`rduckvep_restore_model`
 (`rduckvep_save_model`), [18](#)
`rduckvep_save_model`, [18](#)
`rduckvep_transcript_projection_sql`
 (`rduckvep_builders`), [3](#)