Package ‘nanoarrow’

December 8, 2023

Title Interface to the ‘nanoarrow’ ‘C’ Library
Version 0.3.0.1
Description Provides an ‘R’ interface to the ‘nanoarrow’ ‘C’ library and the ‘Apache Arrow’ application binary interface. Functions to import and export ‘ArrowArray’, ‘ArrowSchema’, and ‘ArrowArrayStream’ ‘C’ structures to and from ‘R’ objects are provided alongside helpers to facilitate zero-copy data transfer among ‘R’ bindings to libraries implementing the ‘Arrow’ ‘C’ data interface.
License Apache License (>= 2)
Encoding UTF-8
RoxygenNote 7.2.3
URL https://github.com/apache/arrow-nanoarrow
BugReports https://github.com/apache/arrow-nanoarrow/issues
Suggests arrow (>= 9.0.0), bit64, blob, hms, rlang, testthat (>= 3.0.0), tibble, vctrs, withr
Config/testthat/edition 3
Config/build/bootstrap TRUE
NeedsCompilation yes
Author Dewey Dunnington [aut, cre] (<https://orcid.org/0000-0002-9415-4582>), Apache Arrow [aut, cph], Apache Software Foundation [cph]
Maintainer Dewey Dunnington <dewey@dunnington.ca>
Repository CRAN
Date/Publication 2023-12-08 04:50:02 UTC

R topics documented:

array_stream_set_finalizer .................................................. 2
as_nanoarrow_array .......................................................... 3
as_nanoarrow_array_stream ................................................ 4
array_stream_set_finalizer

Register an array stream finalizer

Description

In some cases, R functions that return a `nanoarrow_array_stream` may require that the scope of some other object outlive that of the array stream. If there is a need for that object to be released deterministically (e.g., to close open files), you can register a function to run after the stream’s release callback is invoked from the R thread. Note that this finalizer will not be run if the stream’s release callback is invoked from a non-R thread. In this case, the finalizer and its chain of environments will be garbage-collected when `nanoarrow::preserved_empty()` is run.

Usage

`array_stream_set_finalizer(array_stream, finalizer)`

Arguments

array_stream  A `nanoarrow_array_stream`

finalizer  A function that will be called with zero arguments.

Value

array_stream, invisibly
Examples

```r
stream <- basic_array_stream(list(1:5))
array_stream_set_finalizer(stream, function() message("All done!'\))
stream$release()
```

---

**as_nanoarrow_array**  
*Convert an object to a nanoarrow array*

**Description**

In nanoarrow an 'array' refers to the struct `ArrowArray` definition in the Arrow C data interface. At the R level, we attach a `schema` such that functionally the `nanoarrow_array` class can be used in a similar way as an `arrow::Array`. Note that in nanoarrow an `arrow::RecordBatch` and a non-nullable `arrow::StructArray` are represented identically.

**Usage**

```r
as_nanoarrow_array(x, ..., schema = NULL)
```

**Arguments**

- `x`  
  An object to convert to a array
- `...`  
  Passed to S3 methods
- `schema`  
  An optional schema used to enforce conversion to a particular type. Defaults to `infer_nanoarrow_schema()`.

**Value**

An object of class 'nanoarrow_array'

**Examples**

```r
(array <- as_nanoarrow_array(1:5))
as.vector(array)
```

```r
(array <- as_nanoarrow_array(data.frame(x = 1:5)))
as.data.frame(array)
```
as_nanoarrow_array_stream

Convert an object to a nanoarrow array_stream

Description

In nanoarrow, an 'array stream' corresponds to the struct ArrowArrayStream as defined in the Arrow C Stream interface. This object is used to represent a stream of arrays with a common schema. This is similar to an arrow::RecordBatchReader except it can be used to represent a stream of any type (not just record batches). Note that a stream of record batches and a stream of non-nullable struct arrays are represented identically. Also note that array streams are mutable objects and are passed by reference and not by value.

Usage

as_nanoarrow_array_stream(x, ..., schema = NULL)

Arguments

x An object to convert to a array_stream

... Passed to S3 methods

schema An optional schema used to enforce conversion to a particular type. Defaults to infer_nanoarrow_schema().

Value

An object of class 'nanoarrow_array_stream'

Examples

(stream <- as_nanoarrow_array_stream(data.frame(x = 1:5)))
stream$get_schema()
stream$get_next()

# The last batch is returned as NULL
stream$get_next()

# Release the stream
stream$release()
as_nanoarrow_buffer  

Convert an object to a nanoarrow buffer

Description

Convert an object to a nanoarrow buffer

Usage

as_nanoarrow_buffer(x, ...)

Arguments

x  An object to convert to a buffer
...

Passed to S3 methods

Value

An object of class 'nanoarrow_buffer'

Examples

array <- as_nanoarrow_array(c(NA, 1:4))
array$buffers
as.raw(array$buffers[[1]])
as.raw(array$buffers[[2]])
convert_buffer(array$buffers[[1]])
convert_buffer(array$buffers[[2]])

as_nanoarrow_schema  

Convert an object to a nanoarrow schema

Description

In nanoarrow a 'schema' refers to a struct ArrowSchema as defined in the Arrow C Data interface. This data structure can be used to represent an arrow::schema(), an arrow::field(), or an arrow::DataType. Note that in nanoarrow, an arrow::schema() and a non-nullable arrow::struct() are represented identically.
Usage

```r
as_nanoarrow_schema(x, ...)
```

```r
infer_nanoarrow_schema(x, ...)
```

```r
nanoarrow_schema_parse(x, recursive = FALSE)
```

```r
nanoarrow_schema_modify(x, new_values, validate = TRUE)
```

Arguments

- **x**: An object to convert to a schema
- **...**: Passed to S3 methods
- **recursive**: Use `TRUE` to include a children member when parsing schemas.
- **new_values**: New schema component to assign
- **validate**: Use `FALSE` to skip schema validation

Value

An object of class 'nanoarrow_schema'

Examples

```r
infer_nanoarrow_schema(integer())
```

```r
infer_nanoarrow_schema(data.frame(x = integer()))
```

---

**basic_array_stream**

Create ArrayStreams from batches

Description

Create ArrayStreams from batches

Usage

```r
basic_array_stream(batches, schema = NULL, validate = TRUE)
```

Arguments

- **batches**: A list() of `nanoarrow_array` objects or objects that can be coerced via `as_nanoarrow_array()`.
- **schema**: A `nanoarrow_schema` or NULL to guess based on the first schema.
- **validate**: Use `FALSE` to skip the validation step (i.e., if you know that the arrays are valid).

Value

An `nanoarrow_array_stream`
**Examples**

```r
(stream <- basic_array_stream(list(data.frame(a = 1, b = 2))))
as.data.frame(stream$get_next())
stream$get_next()
```

**convert_array**

*Convert an Array into an R vector*

**Description**

Converts array to the type specified by `to`. This is a low-level interface; most users should use `as.data.frame()` or `as.vector()` unless finer-grained control is needed over the conversion. This function is an S3 generic dispatching on `to`: developers may implement their own S3 methods for custom vector types.

**Usage**

```r
convert_array(array, to = NULL, ...)
```

**Arguments**

- `array` A `nanoarrow_array`.
- `to` A target prototype object describing the type to which `array` should be converted, or `NULL` to use the default conversion as returned by `infer_nanoarrow_ptype()`. Alternatively, a function can be passed to perform an alternative calculation of the default ptype as a function of `array` and the default inference of the prototype.
- `...` Passed to S3 methods

**Details**

Conversions are implemented for the following R vector types:

- `logical()`: Any numeric type can be converted to `logical()` in addition to the `bool` type. For numeric types, any non-zero value is considered `TRUE`.
- `integer()`: Any numeric type can be converted to `integer()`; however, a warning will be signaled if the any value is outside the range of the 32-bit integer.
- `double()`: Any numeric type can be converted to `double()`. This conversion currently does not warn for values that may not roundtrip through a floating-point double (e.g., very large uint64 and int64 values).
- `character()`: String and large string types can be converted to `character()`. The conversion does not check for valid UTF-8: if you need finer-grained control over encodings, use `to = blob::blob()`.
\* **factor()**: Dictionary-encoded arrays of strings can be converted to `factor()`; however, this must be specified explicitly (i.e., `convert_array(array, factor())`) because arrays arriving in chunks can have dictionaries that contain different levels. Use `convert_array(array, factor(levels = c(\ldots)))` to materialize an array into a vector with known levels.

- **Date**: Only the `date32` type can be converted to an R `Date` vector.
- **hms::hms()**: Time32 and `time64` types can be converted to `hms::hms()`.
- **difftime()**: Time32, `time64`, and duration types can be converted to R `difftime()` vectors. The value is converted to match the `units()` attribute of `to`.
- **blob::blob()**: String, large string, binary, and large binary types can be converted to `blob::blob()`.
- **vctrs::list_of()**: List, large list, and fixed-size list types can be converted to `vctrs::list_of()`.
- **data.frame()**: Struct types can be converted to `data.frame()`.
- **vctrs::unspecified()**: Any type can be converted to `vctrs::unspecified()`; however, a warning will be raised if any non-null values are encountered.

In addition to the above conversions, a null array may be converted to any target prototype except `data.frame()`. Extension arrays are currently converted as their storage type.

**Value**

An R vector of type `to`.

**Examples**

```r
array <- as_nanoarrow_array(data.frame(x = 1:5))
str(convert_array(array))
str(convert_array(array, to = data.frame(x = double())))
```

---

**convert_array_stream**  
Convert an Array Stream into an R vector

**Description**

Converts `array_stream` to the type specified by `to`. This is a low-level interface; most users should use `as.data.frame()` or `as.vector()` unless finer-grained control is needed over the conversion. See `convert_array()` for details of the conversion process; see `infer_nanoarrow_ptype()` for default inferences of `to`.

**Usage**

```r
convert_array_stream(array_stream, to = NULL, size = NULL, n = Inf)

collect_array_stream(array_stream, n = Inf, schema = NULL, validate = TRUE)
```
infer_nanoarrow_ptype

Arguments

array_stream A nanoarrow_array_stream.

To A target prototype object describing the type to which array should be converted, or NULL to use the default conversion as returned by infer_nanoarrow_ptype(). Alternatively, a function can be passed to perform an alternative calculation of the default ptype as a function of array and the default inference of the prototype.

size The exact size of the output, if known. If specified, slightly more efficient implementation may be used to collect the output.

n The maximum number of batches to pull from the array stream.

schema A nanoarrow_schema or NULL to guess based on the first schema.

validate Use FALSE to skip the validation step (i.e., if you know that the arrays are valid).

Value

• convert_array_stream(): An R vector of type to.
• collect_array_stream(): A list() of nanoarrow_array

Examples

stream <- as_nanoarrow_array_stream(data.frame(x = 1:5))
str(convert_array_stream(stream))
str(convert_array_stream(stream, to = data.frame(x = double())))

stream <- as_nanoarrow_array_stream(data.frame(x = 1:5))
collect_array_stream(stream)

infer_nanoarrow_ptype Infer an R vector prototype

Description

Resolves the default to value to use in convert_array() and convert_array_stream(). The default conversions are:

Usage

infer_nanoarrow_ptype(x)

Arguments

x A nanoarrow_schema, nanoarrow_array, or nanoarrow_array_stream.
Details

- null to `vctrs::unspecified()`
- boolean to `logical()`
- int8, uint8, int16, uint16, and int13 to `integer()`
- uint32, int64, uint64, float, and double to `double()`
- string and large string to `character()`
- struct to `data.frame()`
- binary and large binary to `blob::blob()`
- list, large_list, and fixed_size_list to `vctrs::list_of()`
- time32 and time64 to `hms::hms()`
- duration to `difftime()`
- date32 to `as.Date()`
- timestamp to `as.POSIXct()`

Additional conversions are possible by specifying an explicit value for to. For details of each conversion, see `convert_array()`.

Value

An R vector of zero size describing the target into which the array should be materialized.

Examples

```
infer_nanoarrow_ptype(as_nanoarrow_array(1:10))
```

---

**infer_nanoarrow_ptype_extension**

Implement Arrow extension types

Description

Implement Arrow extension types

Usage

```
infer_nanoarrow_ptype_extension(
  extension_spec,
  x,
  ..., 
  warn_unregistered = TRUE 
)
```

```
convert_array_extension(
```
nanoarrow_array_init

```r
extension_spec,
array,
to,
..., 
warn_unregistered = TRUE
)
```

as_nanoarrow_array_extension(extension_spec, x, ..., schema = NULL)

Arguments

- `extension_spec` An extension specification inheriting from 'nanoarrow_extension_spec'.
- `x, array, to, schema, ...` Passed from `infer_nanoarrow_ptype()`, `convert_array()`, `as_nanoarrow_array()`, and/or `as_nanoarrow_array_stream()`.
- `warn_unregistered` Use FALSE to infer/convert based on the storage type without a warning.

Value

- `infer_nanoarrow_ptype_extension()`: The R vector prototype to be used as the default conversion target.
- `convert_array_extension()` : An R vector of type to.
- `as_nanoarrow_array_extension()`: A nanoarrow_array of type schema.

nanoarrow_array_init Modify nanoarrow arrays

Description

Create a new array or from an existing array, modify one or more parameters. When importing an array from elsewhere, `nanoarrow_array_set_schema()` is useful to attach the data type information to the array (without this information there is little that nanoarrow can do with the array since its content cannot be otherwise interpreted). `nanoarrow_array_modify()` can create a shallow copy and modify various parameters to create a new array, including setting children and buffers recursively. These functions power the `$<-` operator, which can modify one parameter at a time.

Usage

- `nanoarrow_array_init(schema)`
- `nanoarrow_array_set_schema(array, schema, validate = TRUE)`
- `nanoarrow_array_modify(array, new_values, validate = TRUE)`
nanoarrow_buffer_init

Create and modify nanoarrow buffers

Description
Create and modify nanoarrow buffers

Usage
nanoarrow_buffer_init()
nanoarrow_buffer_append(buffer, new_buffer)
convert_buffer(buffer, to = NULL)
Arguments

buffer, new_buffer
	nanoarrow_buffers.

to

A target prototype object describing the type to which array should be converted, or NULL to use the default conversion as returned by infer_nanoarrow_ptype(). Alternatively, a function can be passed to perform an alternative calculation of the default ptype as a function of array and the default inference of the prototype.

Value

• nanoarrow_buffer_init(): An object of class 'nanoarrow_buffer'

• nanoarrow_buffer_append(): Returns buffer, invisibly. Note that buffer is modified in place by reference.

Examples

buffer <- nanoarrow_buffer_init()
nanoarrow_buffer_append(buffer, 1:5)

array <- nanoarrow_array_modify(
  nanoarrow_array_init(na_int32()),
  list(length = 5, buffers = list(NULL, buffer))
) as.vector(array)

describe extension array

Create Arrow extension arrays

Description

Create Arrow extension arrays

Usage

nanoarrow_extension_array(
  storage_array,
  extension_name,
  extension_metadata = NULL
)
nanoarrow_extension_spec

Arguments

storage_array  A `nanoarrow_array`.
extension_name  For `na_extension()`, the extension name. This is typically namespaced separated by dots (e.g., `arrow.r.vctrs`).
extension_metadata  A string or raw vector defining extension metadata. Most Arrow extension types define extension metadata as a JSON object.

Value

A `nanoarrow_array` with attached extension schema.

Examples

```r
nanoarrow_extension_array(1:10, "some_ext", \{
  "key": "value"
\})
```

nanoarrow_extension_spec

`Register Arrow extension types`

Description

Register Arrow extension types

Usage

```r
nanoarrow_extension_spec(data = list(), subclass = character())
register_nanoarrow_extension(extension_name, extension_spec)
unregister_nanoarrow_extension(extension_name)
resolve_nanoarrow_extension(extension_name)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Optional data to include in the extension type specification</td>
</tr>
<tr>
<td>subclass</td>
<td>A subclass for the extension type specification. Extension methods will dispatch on this object.</td>
</tr>
<tr>
<td>extension_name</td>
<td>An Arrow extension type name (e.g., <code>arrow.r.vctrs</code>)</td>
</tr>
<tr>
<td>extension_spec</td>
<td>An extension specification inheriting from <code>nanoarrow_extension_spec</code>.</td>
</tr>
</tbody>
</table>
Value

- `nanoarrow_extension_spec()` returns an object of class 'nanoarrow_extension_spec'.
- `register_nanoarrow_extension()` returns `extension_spec`, invisibly.
- `unregister_nanoarrow_extension()` returns `extension_name`, invisibly.
- `resolve_nanoarrow_extension()` returns an object of class 'nanoarrow_extension_spec' or NULL if the extension type was not registered.

Examples

```r
nanoarrow_extension_spec("mynamespace.mytype", subclass = "mypackage_mytype_spec")
```

---

**nanoarrow_pointer_is_valid**

_Danger zone: low-level pointer operations_

---

**Description**

The `nanoarrow_schema`, `nanoarrow_array`, and `nanoarrow_array_stream` classes are represented in R as external pointers (EXTPTRSXP). When these objects go out of scope (i.e., when they are garbage collected or shortly thereafter), the underlying object's `release()` callback is called if the underlying pointer is non-null and if the `release()` callback is non-null.

**Usage**

```r
nanoarrow_pointer_is_valid(ptr)
nanoarrow_pointer_addr_dbl(ptr)
nanoarrow_pointer_addr_chr(ptr)
nanoarrow_pointer_addr_pretty(ptr)
nanoarrow_pointer_release(ptr)
nanoarrow_pointer_move(ptr_src, ptr_dst)
nanoarrow_pointer_export(ptr_src, ptr_dst)
nanoarrow_allocate_schema()
nanoarrow_allocate_array()
nanoarrow_allocate_array_stream()
nanoarrow_pointer_set_protected(ptr_src, protected)
```
nanoarrow_pointer_is_valid

Arguments

ptr, ptr_src, ptr_dst
  An external pointer to a struct ArrowSchema, struct ArrowArray, or struct ArrowArrayStream.

protected
  An object whose scope must outlive that of ptr. This is useful for array streams since at least two specifications involving the array stream specify that the stream is only valid for the lifecycle of another object (e.g., an AdbcStatement or OGR-Dataset).

Details

When interacting with other C Data Interface implementations, it is important to keep in mind that the R object wrapping these pointers is always passed by reference (because it is an external pointer) and may be referred to by another R object (e.g., an element in a list() or as a variable assigned in a user's environment). When importing a schema, array, or array stream into nanoarrow this is not a problem: the R object takes ownership of the lifecycle and memory is released when the R object is garbage collected. In this case, one can use nanoarrow_pointer_move() where ptr_dst was created using nanoarrow_allocate_*().

The case of exporting is more complicated and as such has a dedicated function, nanoarrow_pointer_export(), that implements different logic schemas, arrays, and array streams:

• Schema objects are (deep) copied such that a fresh copy of the schema is exported and made the responsibility of some other C data interface implementation.

• Array objects are exported as a shell around the original array that preserves a reference to the R object. This ensures that the buffers and children pointed to by the array are not copied and that any references to the original array are not invalidated.

• Array stream objects are moved: the responsibility for the object is transferred to the other C data interface implementation and any references to the original R object are invalidated. Because these objects are mutable, this is typically what you want (i.e., you should not be pulling arrays from a stream accidentally from two places).

If you know the lifecycle of your object (i.e., you created the R object yourself and never passed references to it elsewhere), you can slightly more efficiently call nanoarrow_pointer_move() for all three pointer types.

Value

• nanoarrow_pointer_is_valid() returns TRUE if the pointer is non-null and has a non-null release callback.

• nanoarrow_pointer_addr_dbl() and nanoarrow_pointer_addr_chr() return pointer representations that may be helpful to facilitate moving or exporting nanoarrow objects to other libraries.

• nanoarrow_pointer_addr_pretty() gives a pointer representation suitable for printing or error messages.

• nanoarrow_pointer_release() returns ptr, invisibly.

• nanoarrow_pointer_move() and nanoarrow_pointer_export() return ptr_dst, invisibly.
- `nanoarrow_allocate_array()`, `nanoarrow_allocate_schema()`, and `nanoarrow_allocate_array_stream()` return an array, a schema, and an array stream, respectively.

---

### `nanoarrow_version`

**Description**

Underlying ‘nanoarrow’ C library build

**Usage**

```r
nanoarrow_version(runtime = TRUE)
```

**Arguments**

- `runtime` Compare TRUE and FALSE values to detect a possible ABI mismatch.

**Value**

A string identifying the version of nanoarrow this package was compiled against.

**Examples**

```r
nanoarrow_version()
```

---

### `na_type`

**Create type objects**

**Description**

In nanoarrow, types, fields, and schemas are all represented by a `nanoarrow_schema`. These functions are convenience constructors to create these objects in a readable way. Use `na_type()` to construct types based on the constructor name, which is also the name that prints/is returned by `nanoarrow_schema_parse()`.
Usage

na_type(
    type_name,
    byte_width = NULL,
    unit = NULL,
    timezone = NULL,
    column_types = NULL,
    item_type = NULL,
    key_type = NULL,
    value_type = NULL,
    index_type = NULL,
    ordered = NULL,
    list_size = NULL,
    keys_sorted = NULL,
    storage_type = NULL,
    extension_name = NULL,
    extension_metadata = NULL,
    nullable = NULL
)

na_na(nullable = TRUE)

na_bool(nullable = TRUE)

na_int8(nullable = TRUE)

na_uint8(nullable = TRUE)

na_int16(nullable = TRUE)

na_uint16(nullable = TRUE)

na_int32(nullable = TRUE)

na_uint32(nullable = TRUE)

na_int64(nullable = TRUE)

na_uint64(nullable = TRUE)

na_half_float(nullable = TRUE)

na_float(nullable = TRUE)

na_double(nullable = TRUE)

na_string(nullable = TRUE)
na_large_string(nullable = TRUE)
na_binary(nullable = TRUE)
na_large_binary(nullable = TRUE)
na_fixed_size_binary(byte_width, nullable = TRUE)
na_date32(nullable = TRUE)
na_date64(nullable = TRUE)
na_time32(unit = c("ms", "s"), nullable = TRUE)
na_time64(unit = c("us", "ns"), nullable = TRUE)
na_duration(unit = c("ms", "s", "us", "ns"), nullable = TRUE)
na_interval_months(nullable = TRUE)
na_interval_day_time(nullable = TRUE)
na_interval_month_day_nano(nullable = TRUE)
na_timestamp(unit = c("us", "ns", "s", "ms"), timezone = "", nullable = TRUE)
na_decimal128(precision, scale, nullable = TRUE)
na_decimal256(precision, scale, nullable = TRUE)
na_struct(column_types = list(), nullable = FALSE)
na_sparse_union(column_types = list())
na_dense_union(column_types = list())
na_list(item_type, nullable = TRUE)
na_large_list(item_type, nullable = TRUE)
na_fixed_size_list(item_type, list_size, nullable = TRUE)
na_map(key_type, item_type, keys_sorted = FALSE, nullable = TRUE)
na_dictionary(value_type, index_type = na_int32(), ordered = FALSE)
na_extension(storage_type, extension_name, extension_metadata = "")
Arguments

type_name The name of the type (e.g., "int32"). This form of the constructor is useful for writing tests that loop over many types.

byte_width For na_fixed_size_binary(), the number of bytes occupied by each item.

unit One of 's' (seconds), 'ms' (milliseconds), 'us' (microseconds), or 'ns' (nanoseconds).

timezone A string representing a timezone name. The empty string "" represents a naive point in time (i.e., one that has no associated timezone).

column_types A list() of nanarrow_schemas.

item_type For na_list(), na_large_list(), na_fixed_size_list(), and na_map(), the nanarrow_schema representing the item type.

key_type The nanarrow_schema representing the na_map() key type.

value_type The nanarrow_schema representing the na_dictionary() or na_map() value type.

index_type The nanarrow_schema representing the na_dictionary() index type.

ordered Use TRUE to assert that the order of values in the dictionary are meaningful.

list_size The number of elements in each item in a na_fixed_size_list().

keys_sorted Use TRUE to assert that keys are sorted.

storage_type For na_extension(), the underlying value type.

extension_name For na_extension(), the extension name. This is typically namespaced separated by dots (e.g., arrow.r.vctrs).

extension_metadata A string or raw vector defining extension metadata. Most Arrow extension types define extension metadata as a JSON object.

nullable Use FALSE to assert that this field cannot contain null values.

precision The total number of digits representable by the decimal type

scale The number of digits after the decimal point in a decimal type

Value

A nanarrow_schema

Examples

na_int32()
na_struct(list(col1 = na_int32()))
**Description**

The Arrow format provides a rich type system that can handle most R vector types; however, many R vector types do not roundtrip perfectly through Arrow memory. The vctrs extension type uses `vctrs::vec_data()`, `vctrs::vec_restore()`, and `vctrs::vec_ptype()` in calls to `as.nanoarrow_array()` and `convert_array()` to ensure roundtrip fidelity.

**Usage**

```r
type <- na_vctrs(pctype, storage_type = NULL)
```

**Arguments**

- **ptype**: A vctrs prototype as returned by `vctrs::vec_ptype()`. The prototype can be of arbitrary size, but a zero-size vector is sufficient here.
- **storage_type**: For `na_extension()`, the underlying value type.

**Value**

A `nanoarrow_schema`.

**Examples**

```r
vctr <- as.POSIXlt("2000-01-02 03:45", tz = "UTC")
array <- as.nanoarrow_array(vctr, schema = na_vctrs(vctr))
infer.nanoarrow_ptype(array)
convert_array(array)
```
Index

array, 17
array stream, 17
array_stream_set_finalizer, 2
arrays, 4
arrow::field(), 5
arrow::RecordBatchReader, 4
arrow::schema(), 5
arrow::struct(), 5
as.Date(), 10
as.POSIXct(), 10
as.nanoarrow_array, 3
as.nanoarrow_array(), 6, 11, 21
as.nanoarrow_array_extension
   (infer_nanoarrow_ptype_extension), 10
as.nanoarrow_array_stream, 4
as.nanoarrow_array_stream(), 11
as.nanoarrow_buffer, 5
as.nanoarrow_schema, 5
basic_array_stream, 6
blob::blob(), 8, 10
character(), 7, 10
collect_array_stream
   (convert_array_stream), 8
convert_array, 7
convert_array(), 8–11, 21
convert_array_extension
   (infer_nanoarrow_ptype_extension), 10
convert_array_stream, 8
convert_array_stream(), 9
convert_buffer (nanoarrow_buffer_init), 12
data.frame(), 8, 10
data, 8
difftime(), 8, 10
double(), 7, 10

factor(), 8

hms::hms(), 8, 10

infer_nanoarrow_ptype, 9
infer_nanoarrow_ptype(), 7–9, 11, 13
infer_nanoarrow_ptype_extension, 10
infer_nanoarrow_schema
   (as_nanoarrow_schema), 5
infer_nanoarrow_schema(), 3, 4
integer(), 7, 10

list(), 6
logical(), 7, 10

na_binary (na_type), 17
na_bool (na_type), 17
na_date32 (na_type), 17
na_date64 (na_type), 17
na_decimal128 (na_type), 17
na_decimal256 (na_type), 17
na_dense_union (na_type), 17
na_dictionary (na_type), 17
na_dictionary(), 20
na_double (na_type), 17
na_duration (na_type), 17
na_extension (na_type), 17
na_extension(), 14, 20, 21
na_fixed_size_binary (na_type), 17
na_fixed_size_binary(), 20
na_fixed_size_list (na_type), 17
na_fixed_size_list(), 20
na_float (na_type), 17
na_half_float (na_type), 17
na_int16 (na_type), 17
na_int32 (na_type), 17
na_int64 (na_type), 17
na_int8 (na_type), 17
na_interval_day_time (na_type), 17
na_interval_month_day_nano (na_type), 17
INDEX

na_interval_months (na_type), 17
na_large_binary (na_type), 17
na_large_list (na_type), 17
na_large_list(), 20
na_large_string (na_type), 17
na_list (na_type), 17
na_list(), 20
na_map (na_type), 17
na_map(), 20
na_na (na_type), 17
na_sparse_union (na_type), 17
na_string (na_type), 17
na_struct (na_type), 17
na_time32 (na_type), 17
na_time64 (na_type), 17
na_timestamp (na_type), 17
na_type, 17
na_type(), 17
na_uint16 (na_type), 17
na_uint32 (na_type), 17
na_uint64 (na_type), 17
na_uint8 (na_type), 17
na_vctrs, 21
nanoarrow_allocate_array
  (nanoarrow_pointer_is_valid), 15
nanoarrow_allocate_array_stream
  (nanoarrow_pointer_is_valid), 15
nanoarrow_allocate_schema
  (nanoarrow_pointer_is_valid), 15
nanoarrow_array, 6, 7, 9, 11, 12, 14, 15
nanoarrow_array_init, 11
nanoarrow_array_modify
  (nanoarrow_array_init), 11
nanoarrow_array_set_schema
  (nanoarrow_array_init), 11
nanoarrow_array_stream, 2, 6, 9, 15
nanoarrow_buffer, 13
nanoarrow_buffer_append
  (nanoarrow_buffer_init), 12
nanoarrow_buffer_init, 12
nanoarrow_extension_array, 13
nanoarrow_extension_spec, 14
nanoarrow_pointer_addr_chr
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_add db1
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_addr_pretty
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_export
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_export(), 16
nanoarrow_pointer_is_valid, 15
nanoarrow_pointer_move
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_move(), 16
nanoarrow_pointer_release
  (nanoarrow_pointer_is_valid), 15
nanoarrow_pointer_set_protected
  (nanoarrow_pointer_is_valid), 15
nanoarrow_schema, 6, 9, 12, 15, 17, 20, 21
nanoarrow_schema_modify
  (as_nanoarrow_schema), 5
nanoarrow_schema_parse
  (as_nanoarrow_schema), 5
nanoarrow_schema_parse(), 17
nanoarrow_version, 17
register_nanoarrow_extension
  (nanoarrow_extension_spec), 14
resolve_nanoarrow_extension
  (nanoarrow_extension_spec), 14
schema, 3, 4, 17
units(), 8
unregister_nanoarrow_extension
  (nanoarrow_extension_spec), 14
vctrs::list_of(), 8, 10
vctrs::unspecified(), 8, 10
vctrs::vec_data(), 21
vctrs::vec_ptype(), 21
vctrs::vec_restore(), 21