

Roadmap for Apache Arrow in R

The goal of this document is to identify some broad deliverables for an R interface to the Arrow C++ libraries and other parts of the Arrow ecosystem (such as Spark support, etc.).

This is a public Apache Arrow community document. Please request edit access if you wish to add or make changes

Motivation: Prototypical Initial User Stories

- Send and receive IPC payloads, read Arrow memory from disk

- Read and write open standard columnar files (Parquet, ORC, Feather)

- Interact with Arrow RPC servers (aka Arrow Flight)

- Execute queries using Arrow-native database client

- Arrow-based backend for data frame manipulations

Development tooling, packaging, deployment

- Build tooling, development workflow

- Continuous integration

- Packaging and Deployment

Low-level Memory / IO support

- Buffers and memory pools

- File-like objects

Columnar Format Bindings + R interop

- Arrow columnar data structure APIs

- arrow::DataType (and subclasses)

- arrow::Field

- arrow::Schema

- arrow::Array

- arrow::ChunkedArray

- arrow::Column

- arrow::RecordBatch

- arrow::Table

- R data structure conversions

IPC / Messaging Protocol Support

Data access

- Parquet files

- ORC files

- CSV Files

[HiveServer2 Database Client](#)

[SparkR + sparklyr Integration](#)

[Analytic functions for Arrow columnar data](#)

[Plasma Object Store support](#)

Motivation: Prototypical Initial User Stories

Send and receive IPC payloads, read Arrow memory from disk

Receiving payload use case

- Piece of memory arrives in R through some means, is wrapped in `arrow::Buffer`
- `arrow::Buffer` is interpreted as `arrow::RecordBatch/Table` using `arrow::ipc` utilities
- result either manipulated directly as Arrow data, or converted immediately to native R memory format

Sending payload use case

- R data structure (e.g. `data.frame`) is converted to `arrow::Table`
- `arrow::Table` written to some `arrow::io::OutputStream` using either stream or file protocol
- `OutputStream` is closed, or result of completed write (for `InMemoryOutputStream`) is retrieved as `arrow::Buffer`, to be utilized elsewhere

Read and write open standard columnar files (Parquet, ORC, Feather)

- The user has a single file or collection of files that they wish to read into memory as an R `data.frame`
- A function `read_parquet` can act on a file or directory to read one or more files at once
- Results returned either directly to R `data.frame` or as wrapped `arrow::Table`

Interact with Arrow RPC servers (aka Arrow Flight)

- Connect to RPC server at host and port
- List available dataset
- Request dataset, receive `arrow::Table`

Execute queries using Arrow-native database client

- Connect to server
- Execute SQL query
- Receive results as `arrow::Table`

Arrow-based backend for data frame manipulations

Development tooling, packaging, deployment

Build tooling, development workflow

Source build instructions on:

- Linux, macOS
- Windows

Code style and linting

- Running clang-format, cpplint on R-C++ codebase
- Is there a standard R code tidying tool?

Local unit testing

- Tools in use
- Document procedure for running unit tests

Continuous integration

- Linux builds - Travis CI
- macOS builds - Travis CI
 - This may be optional to avoid making the build matrix too large
- Windows builds

Packaging and Deployment

- Develop multi-platform deployment strategy -- how will users install a complete package on primary Linux distros (RedHat 6 and up, Debian/Ubuntu), macOS, and Windows?
 - Installation should include Arrow libraries with dependencies either bundled or statically linked, and optional packages (Parquet, etc.)
- How to get Arrow into CRAN, work with install.package

Low-level Memory / IO support

Buffers and memory pools

- Interact with arrow::Buffer objects
- Create Buffer from byte-providing R sources
- Wrap arrow::MemoryPool, access default memory pool and see total_bytes_allocated()

File-like objects

We should minimally wrap the file-like objects so that they can return `arrow::Buffer` objects to be used in IPC and other areas

- `io::OSFile` -- regular operating system files
- `MemoryMappedFile`
- `HdfsFile`

Columnar Format Bindings + R interop

The objective is to offer the R user reasonably complete access to the Arrow in-memory columnar format and support for converting between R's native memory representations and the Arrow format.

Arrow columnar data structure APIs

see <http://arrow.apache.org/docs/cpp/namespacedata.html>

`arrow::DataType` (and subclasses)

- Create supported `arrow::DataType`

`arrow::Field`

- Create `arrow::Field`

`arrow::Schema`

A Schema defines the column names and types for a `RecordBatch` or `Table`

- Create `arrow::Schema`

`arrow::Array`

An Array is an atomic columnar data array containing all contiguous / non-chunked memory

- Index single elements in Array
- Call `Slice`
- Call `Array cast`

`arrow::ChunkedArray`

A `ChunkedArray` is a collection of one or more `Array` objects. It is the basic data container used to create `Table` instances. Each Array must have the same type

- Create `arrow::ChunkedArray`
- Call `Cast`
- Call `Slice`

`arrow::Column`

The “Column” data structure is a named `ChunkedArray`

- Create `arrow::Column`
- Call `Cast`

`arrow::RecordBatch`

An ordered string-named sequence of equal-length Arrays

- Create `arrow::RecordBatch`
- Call `Cast` with new Schema
- Call `Slice`

`arrow::Table`

Like `RecordBatch`, but each logical column is a chunked array. Each column does not have to have the same chunking layout as the others.

- Create `arrow::Table`
- Call `Cast` with new Schema
- Call `Slice`

R data structure conversions

- Convert from `arrow::Array` types to R vectors
- Convert from R vectors to `arrow::Array` types in two modes
 - With type inference (`arrow::array(r_vector)`)
 - With explicit type (e.g. `arrow::array(r_vector, type=...)`)
- Decide on conversion API for types unsupported in R
- Convert from `data.frame` to `arrow::RecordBatch` (unchunked) or `arrow::Table` (possibly chunked)
 - With type inference
 - With schema provided
- Convert from `arrow::RecordBatch` or `arrow::Table` to `data.frame`
- Convert to Arrow nested types from nested `data.frame` or `tibble`
 - Including type inference

IPC / Messaging Protocol Support

See <https://github.com/apache/arrow/blob/master/format/IPC.md>

- Read “stream” IPC format
 - see `ipc::RecordBatchStreamReader`
- Read “file” IPC format
 - see `ipc::RecordBatchFileReader`
- Write stream IPC format
 - see `ipc::RecordBatchStreamWriter`
- Write “file” IPC format
 - see `ipc::RecordBatchFileWriter`
- Create individual IPC messages
 - `ipc::SerializeSchema`
 - `ipc::SerializeRecordBatch`

Data access

Parquet files

- Read and write single Parquet files from `arrow::Table`
- Read multiple Parquet files (or a partitioned dataset) a single logical dataset
- Read partitioned dataset with partition-level predicate filters
- Read individual subtrees from a partitioned dataset by partition key (e.g. only read files for single year and month)
- Write non-partitioned or partitioned datasets from R `data.frame` with minimum of code

ORC files

- Provide analogous ORC support to the Parquet support above

CSV Files

- Read CSV files in `RecordBatch` chunks

HiveServer2 Database Client

SparkR + sparklyr Integration

Make available to R users the Arrow-Spark integration that already exists in Spark $\geq 2.3.0$

SparkR Integration

Sparklyr Integration

The relevant work is happening under [sparklyr/pull/1611](https://github.com/sparklyr/sparklyr/pull/1611), so far, we have identified the following API requirements, which are already covered in other sections in this document:

- `arrow::RecordBatch`
- `arrow::Schema`
- IO support with support for `RecordBatchFileWriter`

See also [sparklyr/R/arrow_data.R](https://github.com/sparklyr/R/blob/master/arrow_data.R) for a initial prototype interoperating with Spark using reticulate.

Analytic functions for Arrow columnar data

As there are more of these available, we should define R APIs to invoke them on in-memory Arrow data.

Plasma Object Store support

Create bindings to use the Plasma client to read and write various objects to shared memory managed by Plasma.