DIMSpec Quick Guide - Plumber

Jared M. Ragland*

2023-03-15

The Plumber platform and DIMSpec

This quick guide discusses the application programming interface (API) provided with the Database Infrastructure for Mass Spectrometry (DIMSpec) project. See the project repository on GitHub, it's documentation, or the full DIMSpec User Guide for more details.

The `plumber` package allows code written in R to be made available as a RESTful application programming interface (API). Applications and packages can communicate with the database through the API in a predefined manner without directly connecting to it, or having it available on your local machine. While running DIMSpec in the default standalone manner may not provide much performance improvement, offering selections of the code base through an API service allows a tremendous amount of flexibility for applications, either web based or through the command line, and allows that development to become language agnostic while maintaining a consistent communication layer. With the service running, communication and calculation occur in a background process available either to a local machine or as a network service. As R is a single threaded language, this greatly lowers the overhead and simultaneously makes database interactions more consistent and more performant. This does come at the cost of flexibility, and so the project can be launched in either manner. All of the web applications provided in the project leverage the API for database communication and do not connect directly to the database. This allows launching and hosting of the web applications in separate processes. Basic settings mentioned here are located in the `/config/env_glob.txt` file. The `plumber` API can be launched in a variety of ways:

1. Sourcing the compliance file `source(file.path("R", "compliance.R"))` file with `USE_API` set to `TRUE` will automatically launch the API server in a background process.
2. If the compliance file has been sourced and `USE_API` is set to `FALSE`, run `start_api()` or `reload_api()` to launch using the provided environment settings.
3. Directly launch one of the web applications that requires the API (see the DIMSpec Quick Guide for Web Applications).
4. Directly launch the plumber server file at `/inst/plumber/plumber.R` or, if using RStudio [1], click the "Run API" button with this file loaded in the source pane.

The recommended way to get started with DIMSpec is the first of these options, with `USE_API` set as `TRUE`. By default this will launch the API service in a background process on your local machine and make it available at `http://127.0.0.1:8080`.

*NIST | MML | CSD | jared.ragland@nist.gov

[1] Any mention of commercial products within NIST web pages is for information only; it does not imply recommendation or endorsement by NIST.

Interacting with the API

Communication with a plumber API should use parameters encoded in javascript object notation (JSON) though many endpoints include abstraction parsing for easier use and can handle strings as direct input for many parameters. Endpoints for many predictable read and search interactions are available. Session variables define the connections, and communication and control functions default to those expected values for streamlining (e.g. functions like `api_reload`, `api_open_doc`, and `api_endpoint` may be called without referring explicitly to a session object or URL for the current project).

Interactive documentation is provided by Swagger for each endpoint, allowing users to enter values and get both the return and the URL necessary to execute that endpoint. The main interactivity with the API from an R session or shiny application is through the convenience `api_endpoint` function which was constructed to build an interface that was more amenable to R users. The first argument (i.e. `path`) should always be the endpoint being requested. Additional named parameters are then passed to the API server. For example:

```
api_endpoint(
  path = "compound_data",
  compound_id = 2627,
  return_format = "data.frame"
)
```

issues a request to the API server for the endpoint `compound_data` for `compound_id = 2627` and returns data as a data frame to your session. Endpoints of most use to those using the service will vary according to needs and are detailed in the Plumber section in Technical Details of the [DIMSpec User Guide]. Call them with `api_endpoint(path = *X*)` and any other arguments required by the endpoint. Paths listed here are likely of most use:

- **"__ping"**, **"db__active"**, and **"rdkit__active"** indicate that the server is alive and able communicate with the database and rdkit, respectively;
- **"list__tables"** and **"list__views"** return available tables and views respectively;
- **"compound__data"** and **"peak__data"** return mass spectrometry data associated with a compound or peak and must be called with `compound_id` or `peak_id` equal to the database index of the request; in most cases these should be called with `return_format = "data.frame"`;
- **"table__search"** is a generic database query endpoint analog for `build_db_action` to construct `SELECT` queries and has the most parameters for flexibility; relevant parameters are summarized here:
  - *table_name* should be the name of a single table or view;
  - *column_names* determine which columns are returned;
  - *match_criteria* should be a list of criteria for the search convertible between R lists and JSON as necessary; values should generally follow the convention `list(column_name = value)` and can be nested for further refinement using e.g. `list(column_name = list(value = search_value, exclude = TRUE))` for an exclusion search; when called via `api_endpoint` R objects can be passed programmatically;
  - *and_or* should be either `"AND"` or `"OR"` and determines whether multiple elements of `match_criteria` should be combined in an AND or OR context (e.g. whether `list(column1 = 1, column2 = 2)` should match both or either condition);
  - *limit* is exactly as in the SQL context; leave as `NULL` to return all results or provide a value coercible to an integer to give only that many results;
  - *distinct* is exactly as in the SQL context and should be either `TRUE` or `FALSE`;
  - *get_all_columns* should be either `TRUE` or `FALSE` and will ensure the return of all columns by overriding the `column_names` parameter;
  - *execute* should be either `TRUE` or `FALSE` and determines whether the constructed call results are returned (`TRUE`) or just the URL (`FALSE`); and
  - *single_column_as_vector* should be either `TRUE` or `FALSE` and, if `TRUE`, returns an unnamed vector of results if only a single column is returned.

Once running, the API server is language agnostic, and the same endpoints can be called using any http communication protocol (e.g. `curl`) for application development outside of R. (Note: when using Windows Powershell, by default `curl` is an alias for the cmdlet `Invoke-WebRequest` and uses different parameters than those provided in the Swagger documentation; instead of `curl -X GET *endpoint URI*` use `curl -URI *endpoint URI*`.) These and other endpoints can be easily defined, expanded, or refined as needed to meet project requirements. Use `api_reload` to refresh the server when definitions change, or test interactively prior to deployment using Swagger by launching a separate server either by opening the plumber file and clicking the "Run API" button in RStudio, or using the `api_start` or `api_reload` functions as described elsewhere. To support network deployment, any number of API servers may be launched manually on predefined ports to allow for load balancing.

Modifying the API

The API definition file is located at `/inst/plumber/plumber.R` and is a decorated file of R functions as described in the `plumber` documentation. Endpoints can be added, modified, or removed as needed for specific implementations of DIMSpec. Briefly, plumber endpoints are constructed as are any R functions with the following decorations:

- `@get` determines the endpoint path (e.g. "_ping" in `http://127.0.0.1:8080/_ping`)
- `@param` describes parameters in the function for the Swagger documentation.
- `@serializer` determines the default return data format, by default javascript object notation (JSON).

When modified, any existing plumber service will need to be terminated prior to launching it again. The easiest method once the compliance file has been sourced is to use `api_reload()`, though you may want to use method 4 in the list above for testing a local copy first.