# The JDBC Remote API design

### What is the JDBC Remote API, and why

Normally users will Remote API to use all the functions of the JDBC. Like design their experiment data structure, create new experiment, put and get data, query data.

Below Remote API there is JDBC API. It actually has all the functions needed by users to play with JDBC, and it actually connects to the DB remotely.

SO why do we need a Remote API?

First, the JDBC API the DB used in JDBC is a plain old MongoDB, it cannot be call JDBC alone, and the JDBC API actually implements the functions of the JDBC. So JDBC API is not a client side library, it a server side implementation.

Second, as said above, the JDBC API is a server side implementation, it contains many components that is not concerned by the clients and should not be touched by users like Plug-ins for query and data type. These should be configured by the JDBC administrator not the client, and the client did not want to configure all every time the use JDBC.

Third, when using JDBC with JDBC API, you still need a good amount of JDBC knowledge, which our intended client will not have. Let the Remote API do the dirty work, translates human understandable commands into JDBC API calls.

Forth, the Remote API will provide extra functions like authentication and authorization and even some application specific functions like institute, group or user quota.

Last, the JDBC API is a .NET lib, not .NET user may find it hard to use. Remote API is a non-platform specific API exposed via HTTP or TCP/UDP socket (unlikely). We can provide various client libs for it, and if you need your own it’s very easy to develop one.

## Design：

Implement some of the following RemoteAPI may require modifications to API plug-in even interfaces in CoreServce.

### Use Case Scenario

#### Experiment Management

The API here has little performance requirement, so just use JSON or XML as return body (user negotiate by Accept header)

* Create a signal of a given type with an init string
  + RA11, RA12 (may change the type, datatype and init string from form to query string or support both ways, throw error the query does not return a single experiment)
* Duplicate a signal or experiment to a new location
  + RA13 (same as RA11, with a field to specify duplicate and source id)
* Move a signal or experiment to a new location
  + RA14 (may want to copy RA13 but specify a field to move)
* Delete a signal or experiment
  + RA16
  + Delete the signal or the experiment
  + Defined by a query, if multiple match, delete them all
  + Return the ids of the deleted entity
  + Include a confirm field, if set to false (default) will only return the ids that will be delete but does not delete anything
  + Delete: /entity/{a query to some entity}; query string : confirm=bool, recursive=bool, delete data=(? Do we have this?); return the id deleted or will be deleted and error message
* Return signals or experiments info by a query (without data)
  + Get and return the signal or experiment entity including extra info
  + Get: /entity/{a query to one or more entity}
  + Should be RA17
* Return a signal or experiment Id
  + Like the one above but only returns a set of id
  + Get: /entity/{a query to one or more entity}?onlyids=true
  + RA18
* Set the extra information in a signal or experiment
  + RA15
  + Set key value pairs in extra
  + Support string only at this stage
  + Create new key if not present in the extra
  + Put: /entity/{a query to a signal entity}?extra=true form: extra info
* Modify properties in Signal
  + RA19
  + Like the above but set signal or experiment field not extra
  + Should check if the signal has the certain field (called by users not plug-ins they are allowed to make this mistake)
  + Check by read out the type field (may need to modify the API and Core) and ask the corresponding plug-in to give a Type and use reflect to check it. Or simple let the plug in tells you if this is a valid field
  + The hard part is you may need to use reflect to figure out what’s the correct type for the field and cast the value into that type, or simply let the plug-in do the job
  + Put: /entity/{a query to a signal entity} form: the value

#### Data Read

* Get data with a query
  + RA110
  + Two ways
  + 1. Like RA11 with a data=true Query string Get: /entity/{a query to one with data fragment}?data=true
  + 2. Like RA16 but done in a different controller Get: /data/{a query to one with data fragment} I prefer this way
  + If the data is to big give an error message to suggest using stream
* Stream data from the server to client
  + RA111
  + Get /stream/{a query to one with data fragment}
  + Using protobuf or JSON to serialize, negotiated with user
  + In the stream the first object is always an info object, the plugin and API may need a new method for this
  + How many sample to read out each loop is determined by the sample type (not is has nothing to do with the API or plug-in, this is RemoteAPI level specific)
  + Stop when connection failed
  + Support continue from previous connection (TBC)
* Evaluate an expression
  + TBD

Data Write

* Append data to a signal with a query to specify the signal
  + RA121
  + Post /data/{a query to a single signal with data fragment}
  + Data in body
  + Content type head specifies the body data format
  + Support xml, JSON and protobuf
  + The Remote API is responsible to de-serialize it into sample[]
  + Support blob so no de-serialization and re serialization is needed (TBD)
* Append data to a signal using an id (without call into the query plug-in, may require API support)
  + TBD
* Stream data to server (opens a stream, without reconnection user can stream data to append the signal)
  + RA122
  + The recommended way of upload big data (all fusion data is qualified as big data)
  + Protobuf and JSON is the only way no xml, protobuf is the recommended way
  + Post /stream/{signal query}
  + Support blob (TBD)

#### Authentication and Authorization

TBD

#### Provence

TBD