Server Functional Spec

Release 0.7

Table of Contents

1Schemata Management 2

2Extension Management 3

3API Reference 4

3.1 General 4

3.2 Methods 4

3.2.1 /event 4

3.3 /session 5

3.3.1 GET /session/:id 5

3.3.2 PUT /session/:id 7

4Future 8

# Connections

Connections represent a logically persistent, non-expiring communication channel between a client address space and the server. Client must take measures to avoid connection leak.

Server is configured with a fixed number of allowable connections:

variant.max.connections = 100 // default

After this many active connections, a request for new connection will receive 400 BAD REQUEST will be sent with a more descriptive message in body

When a connection is closed by the client:

* Client is expected to expire all ClientSessions associated with this connection and not to send any requests initiated by objects derived from that connection. If it does, it will receive an internal error.
* Server to expire all ServerSessions associated with this and lets the vacuum thread to clean them out.
* Removes the connection object from the connection table.

When a connection is closed by the server, either due a restart or, in the future, a recreation of the schema,:

* All requests, associated with this connection, will receive a 400 BAD REQUEST with the further text indicating that “connection does not exist.” Upon receiving such response, client is expected to destroy its side of the connection.

# Sessions

# Schemata Management

## General

Schema files are read once during server startup from a file located inside the schema deploy directory. If a fatal error is encountered, the server must be restarted after it is corrected. Until then, all subsequent connection request will receive HTTP status 503 “Service Unavailable.”

Only a single schema file per server, e.g. per deploy directory, is supported at this time.

The schema file must start with the meta section:

{

"meta": {

"name": schema-name::NameString,

"comment": schema-comment::String?

"hooks": [*hook-def:Object*,...]

},

“states”: {...},

“tests”: {...)

}

## Schema deploy directory location

Schema files are located in the OS directory specified by, in the order of significance

* -Dvariant.schemas.dir system property
* variant.data.dir configuration property
* (/schemas classpath directory — future improvement?)

Value is treated the same as Java’s File(String), i.e. if starts with slash is understood as absolute path, otherwise as relative to the applicatioin’s running directory.

# Server Extension API

## Pluggable Event Flushers

As of 0.7.0:

Event flusher class is configured by the variant.event.flusher.class.name and variant.event.flusher.class.init config keys. The class must be on the runtime classpath, e.g. in a jar file inside the /lib directory.

## User Hooks

User hooks are handlers (or listeners) for variant lifecycle events (LSEs). They can be schema, session, state or test scoped.

Schema deployment events:

* State Parsed
* Test Parsed
* Schema Parsed (future)
* Schema Deployed (future)
* Schema Undeployed (future)

Runtime Events:

* Test Qualification
* Test Targeting
* Session Created (future)
* Session Destroyed (future)

### Definition

User hooks are registered in the schema. They must be defined in the meta section and then can be referenced in the subsequent sections.

"meta": {

...

"hooks": [*hook-def:Object*,...]

}

hook-def:= {

“name”: schema-name::NameString,

“class”: qualified-class-name::String

}

Non-meta sections reference a hook by its name, using the hook-refs clause:

test-def := {

...

"hook-refs": [*hook-name::NameString*,...]

}

It is a parse time error for a non-meta section to reference a hook that has not been defined in the meta section.

Schema and session scoped hooks are active at the time of their definition. State and test scoped hooks only apply to a particular state or test and are activated by the hook-refs clause.

### Execution

If more than one hook is registered for an LSE, they form a listener chain and are posted in the ordinal order, i.e. order they are defined. For schema and session scoped hooks, this means the order in which they are mentioned in the “hooks” clause. For the state and test scoped hooks this means the order in which they are mentions in the hook-refs clause of the particular state or test.

For each hook in a chain, if the post() method returns a non-null, the remaining listeners are ignored. Otherwise, the next listener on the chain is posted. If no user hook returned a value, the default hook is posted. Each user LSE type provides a default hook, which is posted if either no user hooks were registered or none returned a value. By contract, default hook will not return null.

# API Reference

## General

* Property names are case insensitive, i.e. createDate is the same as CreateDate.
* Sessions are stored on the server as serialized JSON strings and are deserialized lazily, if server needs them.

## Methods

Notation:

"name": <Number?=NOW>

* **"name":** name to the left of colon is the literal property name. (
* **<Number?=NOW>** information inside the angle brackets is the data type (String/Number/Boolean/Array/Object), optionally followed by the question mark, indicating that this field is optional, optionally followed by the equal sign and the default value. If no question mark, this field is required. If question mark, but no equal sign, the field is optional with no default.
* Note, that the type specification

### /connection

#### POST /connection/:schema-name

Open a new connection to an XDM schema.

|  |  |  |  |
| --- | --- | --- | --- |
| Request Headers | | | |
| Content-Type | text/plain; charset=utf-8 | | |
| Request Body | | | |
| None. | | | |
| Response Headers | | | |
|  | | | |
| Response body | | | |
| {  "id":<String*>,*  "ts":<Number*>,*  "schema":<String>,  } | | | |
| id | | Connection ID. | |
| ts | | Connection creation date, as Epoch time. | |
| schema | | Experiment schema as a JSON string. Must be parsed separately with the schema parser. | |
| Response Codes | | | |
| 200 OK | | | Connection created. |
| 400 BAD\_REQUEST | | | Any application error. |
|  | | |  |

#### DELETE /connect/:schema-id

Close an existing connection to an XDM schema.

|  |  |  |
| --- | --- | --- |
| Request Headers | | |
| Content-Type | text/plain; charset=utf-8 | |
| Request Body | | |
| None. | | |
| Response Headers | | |
|  | | |
| Response body | | |
| None | | |
| Response Codes | | |
| 200 OK | | Connection created. |
| 400 BAD\_REQUEST | | Any application error. |
|  | |  |

### /event

#### POST /event

Trigger a user-defined event in the specified session.

|  |  |  |  |
| --- | --- | --- | --- |
| Request Headers | | | |
| Content-Type | | text/plain; charset=utf-8 | |
| Request Body | | | |
| {  "sid": <String>,  "name": <String>,  "value": <String?>,  "ts": <Number?=NOW>,  "params": <Array?> [  {  "key": <String>,  "val": <String?>  },  ...  ],  } | | | |
| sid | Current variant session ID. | | |
| name | The name of the event. Any string. | | |
| value | Value of the event. Any string. | | |
| ts | Event creation date, as Epoch time. Optional. If not given, defaults to now. | | |
| params | A map of event parameters. Optional. | | |
| Response Headers | | | |
|  | | | |
| Response body | | | |
| None. | | | |
| Response Codes | | | |
| 200 OK | | | Event triggered. |
| 400 BAD\_REQUEST | | |  |

## /session

|  |  |
| --- | --- |
| GET /session/:id | |
| Get session by ID.   |  |  |  | | --- | --- | --- | | Headers | | | | Content-Type | text/plain; charset=utf-8 | | | Request Body | | | | None | | | | Response body | | | | {  "sid": session-id::String,  "ts": timestamp::Number,  "schid": schema-id::String,  "req": request-def::Object ?,  "states": [  {  "state": state-name::String,  "count": visit-count::Number,  },  ...  ],  "tests": [test-name::String,...]  "disqualTests": [test-name::String,...]  }  request-def ::=  {  "state": state-name::String,  "status": status::String,  "comm": is-commited::Boolean,  "params": [  {  "key": param-name::String,  "val": param-value::String  },  ...  ],  "exps": experience-list::List[String]  } | | | | session-id | | Current variant session ID. | | timestamp | | Session creation timestamp, as the Unix Epoch time, i.e. the number of milliseconds since January 1, 1970, 00:00:00 GMT | | schema-id | | Variant schema ID in effect when this session was created. | | state-name | | The name of a visited state. | | visit-count | | Number of times this state has been visited by this session. | | test-name | | The name of a traversed test | | is-qualified | | Is this session qualified for this test. | |  | | | | Response Headers | | | |  | | | | Response Codes | | | | 200 OK | | Session found and returned. | | 400 BAD\_REQUEST | |  |  PUT /session/:id | |
| Save or replace user session by session ID. Idempotent. Body is not parsed but saved in the session store under the ID. Only parsed if required instantiation on Server. |  |

|  |  |  |
| --- | --- | --- |
| Headers | | |
| Content-Type | text/plain; charset=utf-8 | |
| Request Body | | |
| See response body of GET /session/:id | | |
| Response body | | |
| None. | | |
| Response Codes | | |
| 200 OK | | Session created or replaced. |
| 400 BAD\_REQUEST | | Invalid request body. One of a number of user errors was detected in the request body, e.g. a missing required parameter. An additional message will be provided in the HTTP Status header. |

# Future

## Schema Management

* Symbolic variables in schema definition file.
* Multi-line comments.
* Write own parser to provide support for:
  + Line numbers for semantical errors.
  + Preserve original line number before removal of comments.
* Expire connections that appear to have leaked, e.g. have no active sessions for some period of time.
* Hard vs soft schema reload. Hard will recreate an existing schema even if active connections exist, while soft will wait for all active connections to close (possibly indefinitely, i.e. recommended in conjunction with previous point). The goal is to have a mode in which no active sessions will receive a ConnectionClosed exception.