27. Rest

Overview

All of the database and Axon functionality is accessible over a simple REST API based on reading/writing Zinc plaintext files over HTTP. You may use Content Negotiation to access data in alternate formats such as CSV or XML. The SkySpark REST API is a superset of the Project Haystack REST API. SkySpark fully supports all the standardized operations, plus adds in its own vendor specific operations. The full list of operations is specified in Ops chapter, but summarized here:

```
Method     Uri                            Action
--------   -----------------------------  --------------------------
GET        /api/{proj}/rec/{id}           get rec
GET        /api/{proj}/rec/{id}/{tag}     get tag
HAYSTACK   /api/{proj}/about              server summary info
HAYSTACK   /api/{proj}/ops                supported operations
HAYSTACK   /api/{proj}/formats            supported MIME types
HAYSTACK   /api/{proj}/read               read by filer or id
HAYSTACK   /api/{proj}/nav                navigate database for discovery
HAYSTACK   /api/{proj}/watchSub           watch open/subscription
HAYSTACK   /api/{proj}/watchUnsub         watch close/unsubscription
HAYSTACK   /api/{proj}/watchPoll          watch poll
HAYSTACK   /api/{proj}/hisRead            read historized data
HAYSTACK   /api/{proj}/hisWrite           write historized data
HAYSTACK   /api/{proj}/invokeAction       invoke remote action
HAYSTACK   /api/{proj}/pointWrite         read/write point's priority array
HAYSTACK*  /api/{proj}/eval               evaluate axon expr
POST       /api/{proj}/commit             commit list of diffs
SPECIAL    /api/{proj}/export             export view to file
*          /api/{proj}/ext/{name}/*       extension URI space
```

The URIs above assume that the ApiMod is mounted under `/api` which is the default for SkySpark. The operations marked with `HAYSTACK` follow Haystack conventions for sending a

request grid via GET or POST. Operations marked `HAYSTACK*` use Haystack request conventions but are SkySpark extensions to the standardized operations.

APIs

The following APIs may be used with the Haystack and SkySpark REST API:

- `Client`: Fantom client implementation
- Java Haystack Toolkit: open source Java client and server implementation
- haystackExt: SkySpark client connector

Requests

For all operations marked as Haystack compliant, the client makes a request to a server by sending a single grid and the server responses with a grid. Typically grids are encoded using Zinc, but the actual encoding used is pluggable using content negotiation.

GET Requests

Many operations require no grid argument or a grid with a single row. In this case, a request may be performed using a HTTP GET request. The request grid is encoded in the HTTP query string where each query parameter is mapped to tags in a single row. The tag value must be Zinc encoded, otherwise it is assumed to be of a Str type.

Example of request with empty grid:

```
// request URI
/api/demo/about

// request grid in Zinc
ver:"3.0"
empty
```

Example of request with single Str tag:

```
// request URI
/api/demo/read?filter=site

// request grid in Zinc
ver:"3.0"
filter
"site"
```

Example of request with multiple tags encoded as Zinc:

```
// request URI
/api/demo/hisRead?id=@18002d60-a1b6684d&range=yesterday
```

```
// request grid in Zinc
ver:"3.0"
id,range
@18002d60-a1b6684d,"yesterday"
```

POST Requests

If the request grid is anything other than a single row of name/value pairs, then it must be be sent using HTTP POST. The client must encode the grid using a MIME type supported by server. The client can query the supported MIME types using the formats op. The following is an example of posting to the hisRead op using Zinc:

```
POST /api/demo/hisRead HTTP/1.1
Content-Type: text/zinc; charset=utf-8

ver:"3.0"
id,range
@18002d60-a1b6684d,"yesterday"
```

Responses

If the request grid is successfully read by the server, then it processes the operation and returns the HTTP status code 200 and serializes the response result as a MIME encoded grid.

Error Handling

There are three type of errors which may occur when a client makes a request to a server:

- Network I/O errors
- HTTP errors
- Request errors

Network errors occur when the client cannot make a successful TCP connection to the server. This might include invalid host name, port number, or network outage. Typically these sorts of errors are raised as I/O exceptions by the client's runtime.

HTTP errors occur when TCP connections can be successfully established, but the server cannot successfully handle the URI or request grid at the HTTP layer. The following HTTP status codes are used in these cases:

- 400: the client failed to specify a required header such "Content-Type"
- 403: the client is not authorized to access the op
- 404: the URI does not map to a valid operation URI
- 406: the client "Accept" header requested an supported MIME type
- 415: the client posted the request grid using an supported MIME type

- 501: the HTTP method is other than "GET" or "POST"

Request errors occur after the server has successfully read the request grid. If the server cannot fulfill the request for any reason, then it returns HTTP response code of 200 with a error grid as the body. Request errors include but are not limited to:

- invalid request grid data columns
- invalid request grid data types
- unknown or invalid entity identifiers
- inconsistent data types or data values
- server internal errors/exceptions
- Axon evaluation exceptions

Error Grid

If an operation fails after a request grid is successfully read by a server, then the server returns an *error grid*. An error grid is indicated by the presence of the `err` marker tag in the grid metadata. All error grids also include a `dis` tag in the grid metadata with a human readable descripton of the problem. The `meta.errTrace` tag specifies a multi-line String with a full trace of the problem which typically includes:

- Axon call stack
- Fantom call stack

Example of an error grid encoded as Zinc:

```
ver:"3.0" err dis:"Cannot resolve id: badId" errTrace:"UnknownRecErr: badId\n  ...."
empty
```

Content Negotiation

The default for all REST operations is to return the result as a grid (or grids) with the MIME type "text/plain" formatted using Zinc. You can request to receive the results in alternate formats by specifying the "Accept" header in your HTTP request.

The following "Accept" header MIME types are supported:

- Zinc: `text/plain`, `text/zinc`, `*/*`, or unspecified
- Csv: `text/csv`
- Json: `application/json`
- XML: `text/xml`

If you specify an "Accept" header for an unsupported MIME type, then the 406 Unacceptable error code is returned.

Example for reading all site records as CSV:

```
GET /api/demo/eval?expr=readAll(site)
Content-Type: text/plain; charset=utf-8
Accept: text/csv
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv; charset=utf-8

dis,area,geoAddr
Site A,2000ft²,"1000 Main St,Richmond,VA"
Site B,3000ft²,"2000 Cary St,Richmond,VA"
```

« 26. Grids

28. Ops »