Stargate

Representational State Transfer

- A GET to an identifier requests a copy of the information in the supplied content type
- A PUT to an identifier replaces the information
  - The supplied content type determines how it is to be interpreted
- POST adds information
- DELETE eliminates information

Database Operations	HTTP/REST Transaction Equivalents
CREATE	PUT
READ	GET
UPDATE	PUT (replace) or POST (update)
DELETE	DELETE

Notes

- Authentication and access control
  - HTTP Basic Authentication option
  - SSL option
    - Support both client and server side certificate validity checking, including revocation
- Encoding options
  - Octet-stream, XML, and protobuf encoding options are specified
- Table namespace convention
  - Tables are located in a flat namespace
  - To avoid collision, adopt the Java-style naming convention for tables

Resource Identifiers

- RFC 3968 defines general URL syntax:

```
scheme://user:pass@example.net:8080/path/to/file;type=foo?name=val#frag
\_____/  \_______/\___________/\__/\_____/\____/\_____/\________/\___/
   |         |           |       |     |      |       |       |      |
 scheme   userinfo   hostname   port  path  filename param   query fragment
          \_____________________________/
                      authority
```

- The storage systems may be federated across multiple regions and, depending on policy for data exchange, some data items will be only partially replicated
- Hostname in the authority section selects scope
  - DNS in each region can point to the appropriate HA service front end
  - Specific front ends for regional data accesses can also be explicitly selected using more specific DNS names
  - Something like Akamai might be useful once this is scaled up
- Authority section notes
  - Scheme will always be 'http' or 'https'
  - HTTP basic authentication may be employed for access control; however embedding authentication information as 'user:pass' is discouraged

Resource Identifiers

- Data cells are organized into tables, rows and columns
- Cells may contain multiple versions
- Queries and scanners can use an optional timestamp to retrieve the view of the data at the given time (or earlier)

- Addressing for cell or row query (GET)

```
path := '/' <table>
        '/' <row>
        ( '/' ( <column> ( ':' <qualifier> )?
                ( ',' <column> ( ':' <qualifier> )? )+ )?
            ( '/' <timestamp> )? )?
query := ( '?' 'v' '=' <num-versions> )?

/findings/org.someorg
/findings/org.someorg/av
/findings/org.someorg/av:engine1
/findings/org.someorg/av:*/xml?v=10
/findings/org.someorg//2009-04-01.13:42:00              ← not a typo
/findings/org.someorg/av:engine1/2009-04-01.13:42:00?v=1
```

Resource Identifiers

- Addressing for single value store (PUT)
  - Address with table, row, column (and optional qualifier), and optional timestamp

```
path := '/' <table> '/' <row> '/' <column> ( ':' <qualifier> )?
          ( '/' <timestamp> )?

/findings/org.someorg/av
/findings/org.someorg/av:engine1
/findings/org.someorg/av:engine1/2009-04-01.13:42:00
```

- Addressing for multiple (batched) value store (PUT)

```
path := '/' <table> '/' <false-row-key>

/findings/submit
```

Resource Identifiers

- Addressing for row, column, or cell DELETE

```
path := '/' <table>
        '/' <row>
        ( '/' <column> ( ':' <qualifier> )?
          ( '/' <timestamp> )? )?

/findings/org.someorg
/findings/org.someorg/av
/findings/org.someorg/av:engine1
/findings/org.someorg/av:engine1/2009-04-01.13:42:00
```

- Addressing for table creation or schema update (PUT or POST), schema query (GET), or delete (DELETE)

```
path := '/' <table> / 'schema'

/findings/schema
```

Resource Identifiers

- Addressing for scanner creation (POST)

```
path := '/' <table> '/' 'scanner'
```

```
/findings/scanner
```

  - POST body includes scanner specification e.g. table name, start and stop rows, columns, and timestamp range

- Addressing for scanner next item (GET)

```
path := '/' <table> '/' 'scanner' '/' <scanner-id>
```

```
/findings/scanner/112876541342014101a76c3f
```

- Addressing for scanner deletion (DELETE)

```
path := '/' <table> '/' '%scanner' '/' <scanner-id>
```

```
/findings/scanner/112876541342014101a76c3f
```

Query Table List

- `GET /`
  - Retrieves a list of all tables
  - Set Accept header to `text/plain` for plain text output
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful (XML reply)
    - Returns HTTP 200 status
    - Returns an XML entity body containing the list of all available tables
    - Existent but disabled tables are not listed
    - Example:

```
<TableList>
  <table name=”test1”/>
</TableList>
```

Query Table List (cont.)

- If successful (protobufs)
  - Returns HTTP 200 status
  - Existent but disabled tables are not listed
  - Returns the following encoding:

```
message TableList {
  repeated string name = 1;
}
```

Query Table Metadata

- `GET /<table>/regions`
  - Retrieves table region metadata
  - Set Accept header to `text/plain` for plain text output
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful (XML reply)
    - Returns HTTP 200 status
    - Returns an XML entity body containing the table region metadata
    - Example:

```
<TableInfo name="test">
  <Region name="test,,1241574794985" startKey=""
    endKey="39999999" id="1241574794985" />
  <Region name="test,40000000,1279855798549" startKey="40000000"
    endKey="69999999" id="1279855798549" />
  <Region name="test,70000000,1274414944157" startKey="70000000"
    endKey="" id="1274414944157" />
</TableInfo>
```

Query Table Metadata (cont.)

- If successful (protobufs)
  - Returns HTTP 200 status
  - Returns the following encoding:

```
message TableInfo {
  required string name = 1;
  message Region {
    required bytes name = 1;
    optional bytes startKey = 2;
    optional bytes endKey = 3;
    optional int64 id = 4;
    message Location {
      required string host = 1;
      required int32 port = 2;
    }
    optional Location location = 5;
  }
  repeated Region regions = 2;
}
```

Query Table Schema

- `GET /<table>/schema`
  - Retrieves table schema
  - Set Accept header to `text/plain` for plain text output
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful (XML reply)
    - Returns HTTP 200 status
    - Returns an XML entity body containing the table schema
    - Example:

```
<TableSchema name="testtable" IN_MEMORY="false" READONLY="false"
   userattr1="uservalue1">
  <ColumnSchema name="testfamily" COMPRESSION="GZ"
     IN_MEMORY="false" TTL="604800" VERSIONS="1"
     userattr2="uservalue2" />
</TableSchema>
```

Query Table Schema (cont.)

- If successful (protobufs)
  - Returns HTTP 200 status
  - Returns the following encoding:

```
message ColumnSchema {
  optional string name = 1;
  message Attribute {
    required string name = 1;
    required string value = 2;
  }
  repeated Attribute attrs = 2;
  optional int32 ttl = 3;
  optional int32 maxVersions = 4;
  optional string compression = 5;
}

message TableSchema {
  optional string name = 1;
  message Attribute {
    required string name = 1;
    required string value = 2;
  }
  repeated Attribute attrs = 2;
  repeated ColumnSchema columns = 3;
  optional bool inMemory = 4;
  optional bool readOnly = 5;
}
```

Create Table Or Update Table Schema

- `PUT /<table>/schema`
- `POST /<table>/schema`
  - Uploads table schema
    - PUT or POST creates table as necessary
    - PUT fully replaces schema
    - POST modifies schema (add or modify column family)
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 200 status
  - XML schema encoding
    - Set Content-Type header to `text/xml`
    - Supply full table schema for PUT or schema fragment for POST as an XML entity body
      - See Query Table Schema example
  - Protobuf schema encoding
    - Set Content-Type header to `application/x-protobuf`
    - Supply full table schema for PUT or schema fragment for POST
      - See Query Table Schema example

Delete Table

- `DELETE /<table>/schema`
  - Deletes table
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 200 status
  - NOTE: `DELETE /<table>` will not work

Cell Query (Single Value)

- `GET /<table>/<row>/ <column> ( : <qualifier> )? ( / <timestamp> )?`
  - Retrieves one cell, with optional specification of timestamp
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - Set Accept header to `application/octet-stream` for binary
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful (XML reply)
    - Returns HTTP 200 status
    - Returns cell data as an XML entity body
      - Unsafe chars in keys escaped and values in base64
      - Example:

```
<Cell row="testrow" column="test:col1" timestamp="1128765413420141">
    VGhpcyBpcyBhbiBleGFtcGxlIHZhbHVlLgo=
</Cell>
```

Cell Query (Single Value, cont.)

- If successful (protobuf reply)
  - Returns HTTP 200 status
  - Returns cell data using the following encoding:

```
message Cell {
  optional bytes row = 1;
  optional bytes column = 2;
  optional int64 timestamp = 3;
  optional bytes data = 4;
}
```

- If successful (binary reply)
  - Returns HTTP 200 status
  - Returns row, column, and timestamp in X headers: X-Row, X-Column, and X-Timestamp, respectively
  - Returns cell data

Cell or Row Query (Multiple Values)

- `GET /<table>/<row>`
  `( / ( <column> ( : <qualifier> )?`
  `( , <column> ( : <qualifier> )? )+ )?`
  `( / <timestamp> )? )?`
  `( ?v= <num-versions> )?`
  - Retrieves one or more cells from a full row, or one or more specified columns in the row, with optional filtering via timestamp, and an optional restriction on the maximum number of versions to return
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful (XML reply)
    - Returns HTTP 200 status
    - Returns row results as XML entity bodies
      - Unsafe chars in keys escaped and values in base64

```
<CellSet>
  <Cell row="testrow" column="test:c1" timestamp="1128765413420141">
    VGhpcyBpcyBhbiBleGFtcGxlIHZhbHVlLgo=
  </Cell>
</CellSet>
```

Cell or Row Query (Multiple Values, cont.)

- If successful (protobuf reply)
  - Returns HTTP 200 status
  - Returns the following encoding:

```
message Cell {
  optional bytes row = 1;
  optional bytes column = 2;
  optional int64 timestamp = 3;
  optional bytes data = 4;
}

message CellSet {
  repeated Cell values = 1;
}
```

Cell or Row Query (Multiple Values, cont.)

- Suffix Globbing
  - Multiple value queries of a row can optionally append a *suffix glob* on the row key
  - This is a restricted form of scanner which will return all values in all rows that have keys which contain the supplied key on their left hand side, for example:

    org.someorg.*

    → org.someorg.blog

    → org.someorg.home

    → org.someorg.www

    …

Cell Store (Single Value)

- `PUT /<table>/<row>/<column>`
  `( : <qualifier> )?`
  `( / <timestamp> )?`
- `POST /<table>/<row>/<column>`
  `( : <qualifier> )?`
  `( / <timestamp> )?`
  - Stores cell data into the specified location
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 200 status
  - XML option
    - Set Content-Type header to `text/xml`
    - Supply cell value as an XML entity body, see “Cell Query (Single Value)” for example
  - Protobufs option
    - Set Content-Type header to `application/x-protobuf`
    - Supply cell value as protobuf encoding, see “Cell Query (Single Value)” for example

Cell Store (Single Value)

- Binary option
  - Set Content-Type header to `application/octet-stream`
  - Optionally, set “X-Timestamp” header to the desired timestamp
  - Supply cell value as body

Cell Store (Batched)

- `PUT /<table>/<false-row-key>`
- `POST /<table>/<false-row-key>`
  - Use a false row key
  - Row, column, and timestamp values in supplied Cells override the specifications of the same on the path, allowing for posting of multiple values to a table in batch
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 200 status
  - XML encoding option
    - Set Content-Type to `text/xml`
    - Supply commits as XML entity bodies
    - See “Cell or Row Query (Multiple Values)” for example
  - Protobufs option
    - Set Content-Type header to `application/x-protobuf`
    - See “Cell Query (Single Value)” or “Cell or Row Query (Multiple Values)” for KeyValue definition

Row, Column, or Cell Delete

- `DELETE /<table>/<row>`
  `( / ( <column> ( : <qualifier> )?`
  `( / <timestamp> )? )?`
  - Deletes an entire row, a entire column family, or specific cell(s), depending on how specific the data address is
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 200 status
  - NOTE: `DELETE /<table>` will not work

Scanner Creation

- `PUT /<table>/scanner`
- `POST /<table>/scanner`
  - Allocates a new table scanner
  - Use a Cache-Control header directive of “no-cache”
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP 201 status (created)
  - Returns new location URI containing the new scanner ID
    - Example:

      `/table/scanner/112876541342014107c0fa92`
  - XML scanner specification
    - Set Content-Type to `text/xml`
    - Send an XML entity body describing the scanner specification e.g. table name, start and stop rows, columns, and timestamp ranges, for example:

```
<Scanner batch=”1000” startRow=”bar” endRow=”foo”
  columns=”av:,script:,domain:” />
```

Scanner Creation (cont.)

- Protobuf scanner specification
  - Set Content-Type to `application/x-protobuf`
  - Send the following encoding:

```
message Scanner {
  optional bytes startRow = 1;
  optional bytes endRow = 2;
  repeated bytes columns = 3;
  optional int32 batch = 4;
  optional int64 startTime = 5;
  optional int64 endTime = 6;
}
```

Scanner Get Next

- `GET /<table>/scanner/<scanner-id>`
  - Returns the values of the next cells found by the scanner, up to the configured batch amount
  - Set Accept header to `text/xml` for XML reply
  - Set Accept header to `application/x-protobuf` for protobufs
  - Set Accept header to `application/octet-stream` for binary
  - If not successful
    - Returns appropriate HTTP error status code
  - If result is successful but the scanner is exhausted
    - Returns HTTP 204 status (no content)
    - No reply body
  - Otherwise
    - Returns HTTP 200 status
    - See examples for “Cell or Row Query (Multiple Values)”
  - NOTE: Binary option returns only one cell regardless of batching request supplied during scanner creation

Scanner Deletion

- `DELETE /<table>/scanner/<scanner-id>`
  - Deletes resources associated with the scanner
  - This is an optional action; scanners will expire after some globally configurable interval has elapsed with no activity on the scanner
  - If not successful
    - Returns appropriate HTTP error status code
  - If successful
    - Returns HTTP status 200