DataSet REST API

Module Objectives

In this module, you will learn:

- The DataSet REST API
- Interactions with DataSets using HTTP
- Creating and Truncating Tables
- Reading, Writing, Incrementing and Deleting Data

DataSet REST API

The DataSet REST API allows you to interact with Continuuity Reactor Tables (the core DataSets) through HTTP

You can:

- create Tables
- truncate Tables
- read, write, modify and delete data

For DataSets other than Tables, you can only truncate DataSets using this API

Creating a new Table

To create a new table, issue an HTTP PUT method to the URL:

```
PUT <base-url>/tables/<table-name>
```

Parameter

`<table-name>`

Description

Name of the Table to be created

HTTP Responses

Status Code : ***Description***

`200 OK` : The event was successfully received and the Table was either created or already exists
`409 Conflict` : A DataSet of a different type already exists with the given name

---

Creating a new Table: Example

`PUT <base-url>/tables/streams/mytable`

Create a new Table named *mytable*

Comments

- Creates a Table with the name given by `<table-name>`
- Table names should only contain ASCII letters, digits and hyphens
- If a Table with the same name already exists, no error is returned, and the existing Table remains in place
- If a DataSet of a different type exists with the same name—for example, a key/value Table or `KeyValueTable`—this call will return a `409 Conflict` error

---

Writing Data to a Table

To write to a table, send an HTTP PUT method to the table's URI:

```
PUT <base-url>/tables/<table-name>/rows/<row-key>
```

Parameter

`<table-name>`

Description

Name of the Table to be written to

Parameter

`<row-key>`

Description

Row identifier

HTTP Responses

Status Code : *Description*

`200 OK` : The event was successfully received and the Table was successfully written to
`400 Bad Request` : The JSON String map is not well-formed or cannot be parsed as a map from String to String `404 Not Found` : A Table with the given name does not exist

---

Writing Data to a Table: Example

`PUT <base-url>/tables/mytable/rows/status`

Write to the existing Table named *mytable* in a row identified as *status*

Comments

In the body of the request, you must specify the columns and values that you want to write to the Table as a **JSON String map**:

```
{ "x":"y", "y":"a", "z":"1" }
```

This writes three columns named *x*, *y*, and *z* with values *y*, *a*, and *1* respectively

---

Reading Data from a Table

To read data from a Table, address the row that you want to read directly in an HTTP GET method to the table's URI:

```
GET <base-url>/tables/<table-name>/rows/<row-key>[?<column-identifier>]
```

Parameter

`<table-name>`

Description

Name of the Table to be read from

Parameter

`<row-key>`

Description

Row identifier

Parameter

`<column-identifiers>`

Description

An optional combination of attributes and values such as: `start=<column-id> | stop=<column-id> | columns=<column-id>,<column-id>`

HTTP Responses

Status Code : *Description*

`200 OK` : The event was successfully received and the Table was successfully read from `400 Bad Request` : The column list is not well-formed or cannot be parsed `404 Not Found` : A Table with the given name does not exist

---

Reading Data from a Table: Example

`GET <base-url>/tables/mytable/rows/status`

Read from an existing Table named *mytable*, a row identified as *status*

Comments

The response will be a JSON String representing a map from column name to value

Reading the row that was written in the previous example the response is:

```
{"x":"y","y":"a","z":"1"}
```

Reading Data from a Table: Selected Columns

Can specify a list of columns explicitly or give a range of columns

To return only columns *x* and *y*:

```
GET ... /rows/<row-key>?columns=x,y
```

To return all columns equal to or greater than (inclusive) *c5*:

```
GET ... /rows/<row-key>?start=c5
```

To return all columns less than (exclusive, not including) *c5*:

```
GET ... /rows/<row-key>?stop=c5
```

To return all columns equal to or greater than (inclusive) *c2* and less than (exclusive, not including) *c5*:

```
GET .../rows/<row-key>?start=c2&stop=c5
```

Incrementing Data in a Table

To perform an atomic increment of cells of a Table's row, and receive back the incremented values, issue an HTTP POST method to the row's URL:

```
POST <base-url>/tables/<table-name>/rows/<row-key>/increment
```

Parameter

`<table-name>`

Description

Name of the Table to be read from

Parameter

`<row-key>`

Description

Row identifier of row to be read

HTTP Responses

Status Code : *Description*

`200 OK` : The event successfully incremented the row of the Table `400 Bad Request` : The JSON String is not well-formed; or cannot be parsed as a map from a String to a Long; or one of the existing column values is not an 8-byte long value `404 Not Found` : A table with the given name does not exist

---

Incrementing Data in a Table: Example (1 of 2)

`POST <base-url>/streams/mytable/rows/status/increment`

To increment the columns of *mytable*, in a row identified as *status*, by 1

Comments

- In the body of the method, you must specify the columns and values that you want to increment them by as a JSON map from Strings to Long numbers:

```
{ "x": 1, "y": 7 }
```

- Same effect as the corresponding Java Table Increment method
- If successful, the response contains a JSON String map from the column keys to the incremented values.

Incrementing Data in a Table: Example (2 of 2)

For example, if the existing value of column *x* was 4, and column *y* did not exist, then the response would be:

```
{"x":5,"y":7}
```

Column *y* is newly created

Deleting Data from a Table

To delete from a table, submit an HTTP DELETE method:

```
DELETE <base-url>/tables/<table-name>/rows/<row-key>[?<column-identifier>]
```

Parameter

`<table-name>`

Description

Name of the Table to be deleted from

Parameter

`<row-key>`

Description

Row identifier

Parameter

`<column-identifiers>`

Description

An optional combination of attributes and values such as: `start=<column-id> | stop=<column-id> | columns=<column-id>,<column-id>`

HTTP Responses

Status Code : ***Description***

`200 OK` : The event successfully deleted the data of the Table `404 Not Found` : A table with the given name does not exist

---

Deleting Data from a Table: Example

```
GET <base-url>/tables/mytable/rows/status
```

Deletes from an existing Table named *mytable*, a row identified as *status*

Comments

Similarly to *Reading Data from a Table*, explicitly list the columns that you want to delete by adding a parameter of the form `?columns=<column-key,...>`.

Deleting Data from a DataSet

To clear a dataset of all data, submit an HTTP POST request:

```
POST <base-url>/datasets/<dataset-name>/truncate
```

Parameter

`<dataset-name>`

Description

Name of the DataSet to be truncated

HTTP Responses

Status Code : *Description*

`200 OK` : The event successfully deleted the data of the DataSet `404 Not Found` : A DataSet with the given name does not exist

---

Deleting Data from a DataSet: Example

```
POST <base-url>/datasets/mydataset/truncate
```

Deletes all of the data from an existing DataSet named *mydataset*

Comments

This works not only for Tables but with other DataSets, including user-defined Custom DataSets

---

Module Summary

You should now understand the DataSet REST API and:

- Interactions with DataSets using HTTP
- Creating and Truncating Tables
- Reading, Writing, Incrementing and Deleting Data

Module Completed