Table Store

Developer Guide

Developer Guide

Access control

Table Store uses AccessKey and Instance to ensure its access security.

Instance access

When accessing the table data of Table Store, the instance to which the table belongs must first be confirmed. After subscribing to the Table Store service, instances are the logical containers for resource management. Through the Table Store console of Alibaba Cloud, users can create, manage, delete and perform other operations on Table Store instances. After creating an instance, users manage the Table Store’ s tables in the instance. Here, the Instance Name must be unique within an Alibaba Cloud region. Instances with the same name can exist among different Alibaba Cloud regions. The instance name is used as the endpoint prefix:

Public network address: http://instanceName.region.ots.aliyuncs.com

Alibaba Cloud private network address: http://instanceName.region.ots-internal.aliyuncs.com

For example, the access domain names for an instance named myInstance in the China East 1 (hangzhou) region are:

Public network: http://myInstance.cn-hangzhou.ots.aliyuncs.com

Private network: http://myInstance.cn-hangzhou.ots-internal.aliyuncs.com

AccessKey

Table Store authenticates and authorizes requests based on AccessKeys. Therefore, each Table Store request must carry the correct AccessKey information.

Each Alibaba Cloud account can create up to 5 AccessKeys. Users can manage their AccessKeys in the Alibaba Cloud User Center. An AccessKey is composed of an AccessKeyID and an AccessKeySecret.

The AccessKeyID is used to identify the AccessKey and the AccessKeySecret is used to encrypt the Table Store request. The AccessKeySecret is an important credential used to authenticate the request’ s identity. Therefore, users must ensure the confidentiality and security of the AccessKeySecret. The AccessKeys of an Alibaba Cloud account can be used to access all instances of this account.

AccessKeys have two states: Active and Inactive. Only AccessKeys in the Active status can be used to access Table Store. Users can change the Active/Inactive states of their AccessKeys in the Alibaba Cloud User Center. After an AccessKey’ s state is changed, the change will take effect after 1 minute.

Table Store tables

When creating a Table Store table, you must specify the table name, Primary Key and the reserved read/write throughput. In a traditional database, a table has a predefined schema such as the table name, Primary Key, list of its column names and their data types. All rows stored in the table must have the same set of columns. In contrast, Table Store is schema-less NoSQL database, and only requires that a table has a Primary Key, but does not require any definition of other column names and data types in advance. This section mainly introduces the concepts and use of the Table Store tables.

Table name

The table name of Table Store table must comply with the following constraints:

- Composed of English letters (a-z) or (A-Z), numbers (0-9) and underscores (_).
- The first character must be an English letter (a-z), (A-Z) or underscore (_).
- Case sensitive.
- The length must be between 1 to 255 characters.
- The table name must be unique in a single instance, but the same table name can be used in different instances.

Primary Key

When creating a Table Store table, you must specify the table’ s Primary Key. A Primary Key contains at least one, and up to four Primary Key columns. Each Primary Key column has a name and a type.

Table Store has some restrictions on the names and types of the Primary Key columns. For details, refer to the Primary Key section in Table Store Data Model.

Table Store indexes data based on the Primary Key. The Primary Key uniquely identifies each row in the table, so that no two rows can have the same key. The rows are sorted in ascending order by their Primary Key.

Configure the reserved read/write throughput

To ensure the consistent and low-latency performance of Table Store, applications can specify the reserved read/write throughput when creating a table. If the value of the reserved read/write throughput is not 0, Table Store will reserve the necessary resources to meet the provisioned throughput needs. At the same time, bills are generated based on the reserved read/write throughput. Applications can dynamically raise and lower the reserved read/write throughput based on your own business needs. The reserved read/write throughput is set in quantities of read capacity units and write capacity units.

You can update the tables' reserved read/write throughput through the UpdateTable operation. The rules for updating the reserved read/write throughput are as follows:

> There is a required time interval of at least 2 minutes between two updates for the same table. For example, if you update a table's reserved read/write throughput at 12:43 AM, you must then wait until after 12:45 AM to update it for a second time. The required 2-minute time interval between updates is applied at the table level. Between 12:43 AM and 12:45 AM, you can update the reserved read/write throughput for other tables.
>
> There is no frequency limitation to adjust the reserved read/write throughput in a calendar day (00:00:00 to 00:00:00 of the second day in UTC time; 08:00:00 to 08:00:00 of the second day in Beijing time). But the adjustment interval must be more than 2 minutes. Adjusting a table's reserved read/write throughput is defined as adjusting either the read capacity unit or write capacity unit setting. Each such operation is considered as updating the table.
>
> The reserved read/write throughput adjustments will take effect within 1 minute.

During the access to a table, the consumed read/write throughput that exceeds the value of the reserved read/write throughput will be generated to the additional read/write throughput and billed based on the price unit of the additional read/write throughput.

At the early stage, your applications may not have a high throughput. In this case, you can set a low reserved read/write throughput to save cost. As your businesses grow, you may need to raise the tables' reserved read/write throughput to satisfy your business needs. If you want to quickly import a large volume of data right after creating a table, you can set a high reserved write throughput so that the data can be imported quickly. After the data import is completed, you can lower the reserved

read/write throughput.

The data size restrictions of Partition Key

Table Store partitions the table data according to Partition Key ranges. Rows with the same Partition Key will be placed in the same partition. To prevent the large indivisible partitions, it is recommended that the total data size for all rows under a single partition key should not exceed 1 GB.

Table load time

Table Store table will be ready within 1 minute after creation. During the load time, all the read/write data operations for the table will fail. Applications should wait for the table to be loaded before performing any data operations.

Best practices

Table operations

Use Table Store SDK for table operations

Use Table Store Java SDK for table operations

Use Table Store Python SDK for table operations

Data operations

In Table Store, tables are composed of rows. Each row includes Primary Key and Attributes. This chapter will introduce the data operations of Table Store.

Row of Table Store

Row is the basic unit that composes the tables of Table Store. They are composed of a Primary Key and Attributes. A Primary Key is required and all rows in the same table must have the same Primary Key column names and types. Attributes are not mandatory and each row may have different Attributes. For more information, refer to Table Store data model.

There are three types of Table Store data operations:

Single row operations

GetRow — Read a single row from the table.

PutRow — Insert a row into the table. If the row already exists, the existing row will be deleted before the new row is written.

UpdateRow — Update a row. Applications can add or delete the Attribute columns of an existing row or update the value of an existing Attribute column. If this row does not exist, this operation adds a new row.

DeleteRow — Delete a row.

Batch operations

BatchGetRow — Batch read the data of multiple rows in one request.

BatchWriteRow — Batch insert, update or delete multiple rows in one request.

Range read

- GetRange — Read the data of the table within a certain range.

Single row operations

Single row write operations

Table Store has three single row write operations: PutRow, UpdateRow and DeleteRow. The following are the descriptions and considerations of these operations:

PutRow — Write a new row. If this row already exists, the existing row will be deleted before the new row is written.

UpdateRow — Update a row’ s data. Based on the request content, Table Store will add new columns or modify/delete the specified column values for this row. If this row does not exist, a new row will be inserted. However, an UpdateRow request onto an inexistent row with only deletion instructions will not insert a new row.

DeleteRow — Delete a row. If the row to be deleted does not exist, nothing will happen.

By setting the condition field in the request, the application can specify whether a row existence check is performed before executing the write operation. There are three condition checking options:

IGNORE — The row existence checking is not performed.

EXPECT_EXIST — The row is expected to exist. The operation succeeds only if the row exists. Otherwise, the operation fails.

EXPECT_NOT_EXIST — The row is not expected to exist. The operation succeeds only if the row does not exist. Otherwise, the operation fails.

The operation DeleteRow or UpdateRow will fail if the condition checking is EXPECT_NOT_EXIST, because it is meaningless to delete or update the non-existing rows. You can use PutRow operation to update a non-existing row if the condition checking is EXPECT_NOT_EXIST.

Applications will receive the number of the consumed capacity units for successful operations. If the operation fails, such as the parameter check fails, the row' s data size is too large, or the existence check fails, an error code will be returned to the application.

The rules for calculating the number of write capacity units (CU) consumed in each operation are defined as follows:

PutRow — The sum of the data size of Primary Key of the modified row and the data size of Attribute column is divided by 4 KB and rounded up. If the row existence check condition is not IGNORE, a number of read CUs will be consumed which is equivalent to the value rounded up after dividing the data size of the Primary Key of this row by 4 KB. If an operation does not meet the row existence check condition specified by the application, the operation fails and consumes 1 write CU and 1 read CU. For details, refer to API Reference - PutRow.

UpdateRow — The sum of the data size of Primary Key of the modified row and the data size of attribute column is divided by 4 KB and rounded up. If UpdateRow contains an Attribute column which shall be deleted, only the column name is calculated into the data size of this Attribute column. If the row existence check condition is not IGNORE, a number of read CUs will be consumed which is equivalent to the value rounded up after dividing the data size of the Primary Key of this row by 4 KB. If an operation does not meet the row existence check condition specified by the application, the operation fails and consumes 1 write CU and 1 read CU. For details, refer to API Reference - UpdateRow.

DeleteRow — The data size of the Primary Key of the deleted row is divided by 4 KB and rounded up. If the row existence check condition is not IGNORE, a number of read CUs will be consumed which is equivalent to the value rounded up after dividing the data size of the Primary Key of this row by 4 KB. If an operation does not meet the row existence check condition specified by the application, the operation fails and consumes 1 write capacity unit. For details, refer to API Reference - DeleteRow.

A certain number of read CUs will be also consumed by write operations based on the specified condition.

Examples:

The following examples illustrate how the number of write CUs is calculated for single row write operations.

Example 1: Use the PutRow operation to write a row.

```
//PutRow operation
//row_size=len('pk')+len('value1')+len('value2')+8Byte+1300Byte+3000Byte=4322Byte
{
primary_keys:{'pk':1},
attributes:{'value1':String(1300Byte), 'value2':String(3000Byte)}
}

//Original row
//row_size=len('pk')+len('value2')+8Byte+900Byte=916Byte
//row_primarykey_size=len('pk')+8Byte=10Byte
{
primary_keys:{'pk':1},
attributes:{'value2':String(900Byte)}
}
```

The consumption of the read/write CUs for the PutRow operation is described as follows:

When the existence check condition is set to EXPECT_EXIST: The number of write CUs consumed is the value rounded up after dividing 4,322 bytes by 4 KB, and the number of read CUs is the value rounded up after dividing 10 bytes (data size of the Primary Key of the row) by 4 KB. Therefore, the PutRow operation consumes 2 write CUs and 1 read CU.

When the existence check condition is set to IGNORE: The number of write CUs consumed is the value rounded up after dividing 4,322 bytes by 4 KB. No read CU is consumed. Therefore, the PutRow operation consumes 1 write CU and 0 read CU.

When the existence check condition is set to EXPECT_NOT_EXIST: The existence check condition of the specified row fails. The PutRow operation consumes 1 write CU and 1 read CU.

**Example 2: Use the UpdateRow operation to write a new row.

```
//UpdateRow operation
//Length of attribute column deleted will be calculated for row size
//row_size=len('pk')+len('value1')+len('value2')+8Byte+900Byte=922Byte
{
primary_keys:{'pk':1},
attributes:{'value1':String(900Byte), 'value2':Delete}
```

```
}

//The original row does not exist
//row_size=0
```

The consumption of read/write CUs for the UpdateRow operation is described as follows:

When the existence check condition is set to IGNORE: The number of write CUs consumed is the value rounded up after dividing 922 bytes by 4 KB. No read CU is consumed. Therefore, the UpdateRow operation consumes 1 write CU and 0 read CU.

When the existence check condition is set to EXPECT_EXIST: The existence check condition of the specified row fails. The UpdateRow operation consumes 1 write CU and 1 read CU.

Example 3: Use the UpdateRow operation to update an existing row.

```
//UpdateRow operation
//row_size=len('pk')+len('value1')+len('value2')+8Byte+1300Byte+3000Byte=4322Byte
{
primary_keys:{'pk':1},
attributes:{'value1':String(1300Byte), 'value2':String(3000Byte)}
}
//Original row
//row_size=len('pk')+len('value1')+8Byte+900Byte=916Byte
//row_primarykey_size=len('pk')+8Byte=10Byte
{
primary_keys:{'pk':1},
attributes:{'value1':String(900Byte)}
}
```

The consumption of read/write CUs for the UpdateRow operation is described as follows:

When the existence check condition is set to EXPECT_EXIST: The number of write CUs consumed is the value rounded up after dividing 4,322 bytes by 4 KB, and the number of read CUs is the value rounded up after dividing 10 bytes (data size of the Primary Key of the row) by 4 KB. Therefore, the UpdateRow operation consumes 2 write CUs and 1 read CU.

When the existence check condition is set to IGNORE: The number of write CUs consumed is the value rounded up after dividing 4,322 bytes by 4 KB. No read CU is consumed. Therefore, the UpdateRow operation consumes 1 write CU and 0 read CU.

Example 4: Use the DeleteRow operation to delete a non-existent row.

```
//The original row does not exist
//row_size=0
```

```
//DeleteRow operation
//row_size=0
//row_primarykey_size=len('pk')+8Byte=10Byte
{
primary_keys:{'pk':1},
}
```

The data size both before and after modification is 0. Whether the DeleteRow operation succeeds or fails, 1 write CU at least is consumed. Therefore, this DeleteRow operation consumes 1 write CU.

The consumption of read/write CUs for the DeleteRow operation is described as follows:

> When the existence check condition is set to IGNORE: The number of write CUs consumed is the value rounded up after dividing 10 bytes (data size of the Primary Key of the row) by 4 KB, and the number of read CUs is the value rounded up after dividing 10 bytes by 4 KB. Therefore, the DeleteRow operation consumes 1 write CU and 1 read CU.
>
> When the existence check condition is set to EXPECT_EXIST: The number of write CUs consumed is the value rounded up after dividing 10 bytes (data size of the Primary Key of the row) by 4 KB. No read CU is consumed. Therefore, the DeleteRow operation consumes 1 write CU and 0 read CU.

For more information, refer to the PutRow, UpdateRow and DeleteRow chapters in the API Reference.

Single row read operations

There is only one single row read operation: GetRow.

Applications provide the complete Primary Key and the names of all columns to be returned. The column names can be either Primary Key or Attribute columns. Users may not specify any column names to return, in which case all data of the row data will be returned.

Table Store calculates the consumed read CUs by adding the data size of the Primary Key of the read row and the data size of the read Attribute column. The data size is divided by 4KB and rounded up as the number of read CUs consumed in this read operation. If the specified row does not exist, 1 read CU is consumed. Single row read operations do not consume write CUs.

Example:

This example illustrates how the number of read capacity units consumed by a GetRow operation is calculated:

```
//GetRow operation
//row_size=len('pk')+len('value1')+len('value2')+8Byte+1200Byte+3100Byte=4322Byte
{
primary_keys:{'pk':1},
```

```
attributes:{'value1':String(1200Byte), 'value2':String(3100Byte)}
}

//GetRow operation
//Reading data size=len('pk')+len('value1')+8Byte+1200Byte=1216Byte
{
primary_keys:{'pk':1},
columns_to_get:{'value1'}
}
```

The number of consumed read capacity units is rounded up after dividing 1218 Bytes by 4KB. This GetRow operation consumes 1 read CU.

For more information, refer to the GetRow chapter in the API Reference.

Multi-Row operations

Table Store provides two multi-row operations: BatchWriteRow and BatchGetRow.

BatchWriteRow operations are used to insert, modify or delete multiple rows from one or more tables. BatchWriteRow operations can be considered as a batch of multiple PutRow, UpdateRow and DeleteRow operations. The sub-operations in a single BatchWriteRow are executed independently. Table Store will return the execution results for each sub-operation to the application separately. It may be the case that parts of the request succeed while other parts fail. Even if an error is not returned for the overall request, the application still must check the return results for each sub-operation to determine the correct status. The write CUs consumed by each BatchWriteRow sub-operation are calculated independently.

BatchGetRow is used to read multiple rows from one or more tables. In BatchGetRow, each sub-operation is executed independently. Table Store will return the execution results for each sub-operation to the application separately. It may be the case that parts of the request succeed while other parts fail. Even if an error is not returned for the overall request, the application still must check the return results for each sub-operation to determine the correct statuses. The read capacity units consumed by each BatchGetRow sub-operation are calculated independently.

For more information, refer to the BatchWriteRow and BatchGetRow chapters in the API Reference.

Range read operations

Table Store provides the range read operation GetRange. This operation returns data in the specified range of Primary Key to applications.

The rows in Table Store tables are sorted in the ascending order of Primary Keys. Each GetRange operation specifies a left-closed-right-open range, and returns data from rows with Primary Keys in this range. End points of ranges are composed of either effective Primary Keys or the virtual points: INF_MIN and INF_MAX. The number of columns for the virtual point must be the same as for the Primary Key. Here, INF_MIN represents an infinitely small value, so any values of other types are

strictly greater than it. INF_MAX represents an infinitely large value, so any values of other types are strictly smaller than it.

GetRange operations must specify the columns to get by their names. The request column name can contain multiple column names. If a row has a Primary Key in the read range but does not contain other column specified for return, the results returned by the request will not contain data from this row. If columns to get are not specified, the entire rows will be returned.

GetRange operations must specify the read direction which can be either forward or backward. If a table has two Primary Key columns A and B and A<B : When [A, B) is read in the forward direction, rows with Primary Key greater than or equal to A and less than B will be returned in the order A to B. When [B,A) is read in the backward direction, rows greater than A and less than or equal to B are retuned in the order B to A.

GetRange operations can specify the maximal number of returned rows. Table Store will end the operation as soon as the maximal number of rows is returned according to the forward or backward direction, even if there are remaining rows in the specified ranges.

In the following situations, GetRange operations may stop execution and return data to the application:

The total size of row data to return reaches 4 MB.

The number of rows to return is equal to 5000.

The number of returned rows is equal to the maximal number of rows specified in requests to be returned.

In such premature-return cases, responses returned by the GetRange request will contain the Primary Key for the next row of unread data. Applications can use this value as the starting point for a subsequent GetRange operation. If the Primary Key for the next unread row is null, this indicates all data in the read range has been returned.

Table Store accumulates the total data size of Primary Key and the Attribute column read for all rows from the read range start point to the next row of unread data. The data size is then divided by 4 KB and rounded up to find the number of the consumed read CUs. For example, if the read range contains 10 rows and the Primary Key and the actual data size of the Attribute column read for each row is 330 Bytes, the number of consumed read CU is 1 (divide the total read data size 3.3 KB by 4 KB and rounded up to 1).

Examples:

The following examples illustrate how the number of read CUs is calculated for the GetRange operations.

In these examples, the table contents are as follows. PK1 and PK2 are the table’ s Primary Key

columns, and their types are String and Integer respectively. A and B are the table’ s attribute columns.

PK1	PK2	Attr1	Attr2
‘A’	2	‘Hell’	‘Bell’
‘A’	5	‘Hello’	Non-exist
‘A’	6	Non-exist	‘Blood’
‘B’	10	‘Apple’	Non-exist
‘C’	1	Non-exist	Non-exist
‘C’	9	‘Alpha’	Non-exist

Example 1: Read Data in a specified range.

```
//Request
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "A"), ("PK2", INTEGER, 2)
exclusive_end_primary_key: ("PK1", STRING, "C"), ("PK2", INTEGER, 1)

//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 2)
attribute_columns:("Attr1", STRING, "Hell"), ("Attr2", STRING, "Bell")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 5)
attribute_columns:("Attr1", STRING, "Hello")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 6)
attribute_columns:("Attr2", STRING, "Blood")
} ,
{
primary_key_columns:("PK1", STRING, "B"), ("PK2", INTEGER, 10)
attribute_columns:("Attr1", STRING, "Apple")
}
}
```

Example 2: Use INF_MIN and INF_MAX to read all data in a table.

```
//Request
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", INF_MIN)
exclusive_end_primary_key: ("PK1", INF_MAX)
```

```
//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 2)
attribute_columns:("Attr1", STRING, "Hell"), ("Attr2", STRING, "Bell")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 5)
attribute_columns:("Attr1", STRING, "Hello")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 6)
attribute_columns:("Attr2", STRING, "Blood")
} ,
{
primary_key_columns:("PK1", STRING, "B"), ("PK2", INTEGER, 10)
attribute_columns:("Attr1", STRING, "Apple")
}
{
primary_key_columns:("PK1", STRING, "C"), ("PK2", INTEGER, 1)
}
{
primary_key_columns:("PK1", STRING, "C"), ("PK2", INTEGER, 9)
attribute_columns:("Attr1", STRING, "Alpha")
}
}
```

Example 3: Use INF_MIN and INF_MAX in certain primary key columns.

```
//Request
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "A"), ("PK2", INF_MIN)
exclusive_end_primary_key: ("PK1", STRING, "A"), ("PK2", INF_MAX)

//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 2)
attribute_columns:("Attr1", STRING, "Hell"), ("Attr2", STRING, "Bell")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 5)
attribute_columns:("Attr1", STRING, "Hello")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 6)
attribute_columns:("Attr2", STRING, "Blood")
}
}
```

Example 4: Backward reading.

```
//Request
table_name: "table_name"
direction: BACKWARD
inclusive_start_primary_key: ("PK1", STRING, "C"), ("PK2", INTEGER, 1)
exclusive_end_primary_key: ("PK1", STRING, "A"), ("PK2", INTEGER, 5)

//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "C"), ("PK2", INTEGER, 1)
},
{
primary_key_columns:("PK1", STRING, "B"), ("PK2", INTEGER, 10)
attribute_columns:("Attr1", STRING, "Apple")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 6)
attribute_columns:("Attr2", STRING, "Blood")
}
}
```

Example 5: Specify a column name not including a PK.

```
//Request
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "C"), ("PK2", INF_MIN)
exclusive_end_primary_key: ("PK1", STRING, "C"), ("PK2", INF_MAX)
columns_to_get: "Attr1"

//Response
cosumed_read_capacity_unit: 1
rows: {
{
attribute_columns: {"Attr1", STRING, "Alpha"}
}
}
```

Example 6: Specify a column name including a PK.

```
//Request
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "C"), ("PK2", INF_MIN)
exclusive_end_primary_key: ("PK1", STRING, "C"), ("PK2", INF_MAX)
columns_to_get: "Attr1", "PK1"

//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "C")
```

```
}
{
primary_key_columns:("PK1", STRING, "C")
attribute_columns:("Attr1", STRING, "Alpha")
}
}
```

Example 7: Use limit and breakpoints.

```
//Request 1
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "A"), ("PK2", INF_MIN)
exclusive_end_primary_key: ("PK1", STRING, "A"), ("PK2", INF_MAX)
limit: 2

//Response 1
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 2)
attribute_columns:("Attr1", STRING, "Hell"), ("Attr2", STRING, "Bell")
},
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 5)
attribute_columns:("Attr1", STRING, "Hello")
}
}
next_start_primary_key:("PK1", STRING, "A"), ("PK2", INTEGER, 6)

//Request 2
table_name: "table_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", STRING, "A"), ("PK2", INTEGER, 6)
exclusive_end_primary_key: ("PK1", STRING, "A"), ("PK2", INF_MAX)
limit: 2

//Response 2
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", STRING, "A"), ("PK2", INTEGER, 6)
attribute_columns:("Attr2", STRING, "Blood")
}
}
```

Example 8: Use the GetRange operation to calculate the consumed read CUs.

GetRange is performed on the following table. PK1 is the table’s Primary Key column and Attr1 and Attr2 are the Attribute columns.

PK1	Attr1	Attr2
1	Non-existent	String(1000Byte)
2	8	String(1000Byte)

3	String(1000Byte)	Non-existent
4	String(1000Byte)	String(1000Byte)

```
//Request
table_name: "table2_name"
direction: FORWARD
inclusive_start_primary_key: ("PK1", INTEGER, 1)
exclusive_end_primary_key: ("PK1", INTEGER, 4)
columns_to_get: "PK1", "Attr1"

//Response
cosumed_read_capacity_unit: 1
rows: {
{
primary_key_columns:("PK1", INTEGER, 1)
},
{
primary_key_columns:("PK1", INTEGER, 2),
attribute_columns:("Attr1", INTEGER, 8)
},
{
primary_key_columns:("PK1", INTEGER, 3),
attribute_columns:("Attr1", STRING, String(1000Byte))
},
}
```

For this GetRange request:

Data size of the first row: len（‘PK1’）+ 8 Bytes = 11 Bytes

Data size of the second row: len（‘PK1’）+ 8 Bytes + len（‘Attr1’）+ 8 Bytes = 24 Bytes

Data size of the third row: len（‘PK1’）+ 8 Bytes + len（‘Attr1’）+ 1000 Bytes = 1016 Bytes

The number of the consumed read CUs is the value rounded up after dividing 1051 Bytes (11 Bytes + 24 Bytes + 1016 Bytes) by 4KB. So this GetRange operation consumes 1 read CU.

For more information, refer to the GetRange chapter in the API Reference.

Best practices

Data operations

Table Store SDK for data operation

TableStore Java SDK for data operation

TableStore Python SDK for data operation

Table Store API

Applications can use the Table Store SDK officially released by Alibaba Cloud to access Table Store. They can also use the POST method to send HTTP requests to access Table Store.

This chapter will introduce the HTTP request structure and data format, and explain how to structure HTTP requests and parse their return results. Finally, it will explain the error status codes returned by Table Store requests. Developers using Java or Python can use the official Java and Python SDKs. If you need to use a language other than Java or Python to access Table Store, you can use HTTP messages to interact with Table Store based on the content in this chapter, or independently compile the SDK by yourself.

Current API version

API version: 2014-08-08

HTTP messages

Table Store accepts HTTP requests from applications, processes them according to the relevant logic and uses HTTP messages to return results data after processing. Data in the HTTP requests and responses are organized using ProtocolBuffer protocol format. For more information on ProtocolBuffer protocol, refer to Table Store ProtocolBuffer Message Definitions. In the following parts, we will introduce the specific formats of the HTTP request headers and bodies as well as responses.

HTTP request

HTTP request URL

The URLs used to access Table Store are structured in the manner below:

```
Internet access URL: http://<instance>.<region>.ots.aliyuncs.com/<operation>

Intranet access URL: http://<instance>.<region>.ots-internal.aliyuncs.com/<operation>
```

> instance — The instance name. Instances are created by users. The information for instances of a cloud account can be viewed on the Table Store console. Not case sensitive.

region — The Alibaba Cloud service node. The Table Store service is deployed across different Alibaba Cloud service nodes located at multiple regions. When creating an instance, you must specify an Alibaba Cloud region. You can view the Alibaba Cloud region names which the instances belong to in the Table Store console. Not case sensitive.

operation — The Table Store operation name. All Table Store operations are listed in the API Reference chapter. Case sensitive.

The following URL is the destination URL for a ListTable request targeting an instance named myInstance at China East 1 (hangzhou) region.

```
http://myInstance.cn-hangzhou.ots.aliyuncs.com/ListTable
```

HTTP request header

Table Store requires that the HTTP request headers contain the following information:

x-ots-date — The request issue time. The date uses the rfc822 standard and UTC time. Format: "%a, %d %b %Y %H:%M:%S GMT".

x-ots-apiversion — The API version number. Version numbers are date strings. This document uses the API version number 2014-08-08.

x-ots-accesskeyid — The user's AccessKeyID.

x-ots-instancename — The instance name.

x-ots-contentmd5 — The MD5 for the HTTP body, encoded by base64.

x-ots-signature — The request signature. The signature calculation method is as follows:

```
  Signature = base64(HmacSha1(AccessKeySecret, StringToSign));

StringToSign = CanonicalURI + '\n' + HTTPRequestMethod + '\n\n' + CanonicalHeaders

CanonicalHeaders = LowerCase (HeaderName1) + ':' + Trim(HeaderValue1) + '\n' + ... + LowerCase
(HeaderNameN) + ':' + Trim(HeaderValueN) + '\n'
```

Descriptions of the functions used in the above pseudo-code:

HmacSha1 — Hmac-Sha1 encryption algorithm. During Table Store request

signature calculation, use StringToSign as the message and AccessKeySecret as the secret key.

base64 — Base64 encoding algorithm.

LowerCase — Convert all letters in the string to lowercase.

Trim — Remove spaces at the front and end of the string.

CanonicalURI — The path section in the HTTP URL is given in the example below, with the CanonicalURI set as /ListTable.

```
http://myInstance.cn-hangzhou.ots.aliyuncs.com/ListTable
```

HTTPRequestMethod — HTTP request methods (such as GET, POST or PUT). the Table Store HTTP API only supports the POST method. Note that POST must be written in uppercase.

CanonicalHeaders — CanonicalHeaders are Table Store HTTP header strings (not include the x-ots-signature header) structured according to the following rules:

It must contain and only contain the Table Store standard headers that begin with 'x-ots- '.

All the header item names must be in lowercase and the values must use the Trim operation to remove the spaces.

The header items are ordered in ascending lexicographic order by name.

The header items' names and values must be separated by ':' .

A newline character '\n' must be used to separate headers.

Table Store will verify the following HTTP requests:

The consistency of the x-ots-contentmd5 header value with the MD5 calculated for the data contained in the HTTP body.

The signature in the request header is correct.

The time in x-ots-date differs by less than 15 minutes from the server time.

If the verification request fails, Table Store will return an authentication error.

HTTP request body

Table Store requires that the HTTP request Body is a string serialized by the ProtocolBuffer message defined by Table Store. The body length should be less than 2 MB.

For Table Store requests' ProtocolBuffer message definitions, refer to Table Store ProtocolBuffer message definitions.

HTTP response

HTTP response header

Table Store will return the HTTP response with the following headers items:

x-ots-date — The request issue time. The date uses the rfc822 standard and UTC time. Format: "%a, %d %b %Y %H:%M:%S GMT".

x-ots-requestid — The request ID of this request.

x-ots-contenttype — The response content type. Fixed as "protocol buffer" string.

x-ots-contentmd5 — An MD5 value calculated based on the response content and encoded by base64.

Authorization — The response signature. The response will contain a signature only if the request signature has passed the Table Store verification. The signature calculation method is as follows:

```
Authorization = 'OTS ' + AccessKeyID + ':' + base64 (HmacSha1(AccessKeySecret, stringToSign))

StringToSign = CanonicalHeaders + CanonicalURI

CanonicalHeaders = LowerCase (HeaderName1) + ':' + Trim(HeaderValue1) + '\n' + ... + LowerCase (HeaderNameN) + ':' + Trim(HeaderValueN) + '\n'
```

Descriptions of the functions used in the above pseudo-code :

The functions are the same as those used in the request above.

CanonicalURI — The path section in the HTTP URL is given in the following example, with the CanonicalURI set as /ListTable.

```
http://myInstance.cn-hangzhou.ots.aliyuncs.com/ListTable
```

CanonicalHeaders — The Table Store HTTP header strings (not including the x-ots-signature header) structured according to the following rules:

It must contain and only contain the Table Store standard headers that begin with 'x-ots- '.

All the header item names must be in lowercase and the values must use the Trim operation to remove the spaces.

The header items are ordered in the ascending lexicographic order by name.

The header items' names and values must be separated by ':'.

A newline character must be used to separate headers.

The client should verify the Table Store response as follows:

Verify that the signature in the response header is correct.

Verify that the time in x-ots-date differs by less than 15 minutes from the client's time (positive or negative).

Verify the consistency of the x-ots-contentmd5 header value with the MD5 calculated for the response data.

If the verification response fails, users should reject the response data in code as this response may not come from the Table Store service.

HTTP response content

Table Store requires that the HTTP response content is a string that has been serialized by the ProtocolBuffer message defined by Table Store. The length of Body cannot exceed 2 MB. Each Table Store request message corresponds to one Table Store response message. After the application deserializes the response content, it can read the Table Store operation results.

Signature examples

Two request and response signature verification examples are provided below. Following the implementation of a signature algorithm, users can use the following examples to test if the algorithm was implemented correctly.

Request signature examples

Assume that the user’ s AccessKeyID is ‘29j2NtzlUr8hjP8b’ and the AccessKeySecret is ‘8AKqXmNBkl85QK70cAOuH4bBd3gS0J’ .

```
POST /ListTable HTTP/1.0
x-ots-date: Tue, 12 Aug 2014 10:23:03 GMT
x-ots-apiversion:2014-08-08
x-ots-accesskeyid: 29j2NtzlUr8hjP8b
x-ots-contentmd5: 1B2M2Y8AsgTpgAmY7PhCfg==
x-ots-instancename: naketest
```

Thus, the user’ s request signature result will be as follows<!— Zhao Feng, please confirm this is correct during review—>

```
stringToSign = '/ListTable\nPOST\n\nx-ots-accesskeyid:29j2NtzlUr8hjP8b\nx-ots-apiversion:2014-08-08\nx-ots-contentmd5:1B2M2Y8AsgTpgAmY7PhCfg==\nx-ots-date:Tue, 12 Aug 2014 10:23:03 GMT\nx-ots-instancename:naketest\n'

signature = base64(HmacSha1('8AKqXmNBkl85QK70cAOuH4bBd3gS0J', stringToSign))
= '4xap392B7EBpN+RmlHgNowjoG1w='
```

Response signature examples

Assume that the user’ s AccessKeyID is ‘29j2NtzlUr8hjP8b’ and the AccessKeySecret is ‘AKqXmNBkl85QK70cAOuH4bBd3gS0J’ .

```
/ListTable
x-ots-contentmd5: 1B2M2Y8AsgTpgAmY7PhCfg==
x-ots-requestid: 0005006c-0e81-db74-4a34-ce0a5df229a1
x-ots-contenttype: protocol buffer
x-ots-date:Tue, 12 Aug 2014 10:23:03 GMT
```

Thus, the Table Store response signature result will be as follows:

```
stringToSign = 'x-ots-contentmd5:1B2M2Y8AsgTpgAmY7PhCfg==\nx-ots-contenttype:protocol buffer\nx-ots-date:Tue, 12 Aug 2014 10:23:03 GMT\nx-ots-requestid:0005006c-0e81-db74-4a34-ce0a5df229a1\n/ListTable'

authorization = 'OTS ' + AccessKeyID + ':' + base64(HmacSha1('8AKqXmNBkl85QK70cAOuH4bBd3gS0J', stringToSign))
```

Error messages

This section lists the error types, description messages and HTTP status codes for all possible errors in Table Store API.

The table below lists all the possible error messages returned by Table Store. Some errors may be resolved by retrying the operation. These error messages have a value of **Yes** in the **Retry** column. Other messages have **No**.

Permission verification errors

HTTPStatus	ErrorCode	ErrorMsg	Description	Retry
403	OTSAuthFailed	The AccessKeyID does not exist.	The AccessKeyID does not exist.	No
403	OTSAuthFailed	The AccessKeyID is disabled.	The AccessKeyID is disabled.	No
403	OTSAuthFailed	The user does not exist.	This user does not exist.	No
403	OTSAuthFailed	The instance is not found.	This instance does not exist.	No
403	OTSAuthFailed	The user has no privilege to access the instance.	You do not have permission to access this instance.	No
403	OTSAuthFailed	The instance is not running.	This instance is not in the Running status.	No
403	OTSAuthFailed	The user has no privilege to access the instance.	You do not have permission to access this instance.	No
403	OTSAuthFailed	Signature mismatch.	The signature does not match.	No
403	OTSAuthFailed	Mismatch between system time and x-ots-date: { Date }.	The server time that differs from the x-ots-date time in the header is more than the	No

			allowed range.	

HTTP message errors

HTTPStatus	ErrorCode	ErrorMsg	Description	Retry
413	OTSRequestBodyTooLarge	The size of POST data is too large.	The data volume sent in the user’ s POST request is too large.	No
408	OTSRequestTimeout	Request timeout.	The client took too long to complete the request.	No
405	OTSMethodNotAllowed	OTSMethodNotAllowedOnly POST method for requests is supported.	Only supports POST requests.	No
403	OTSAuthFailed	Mismatch between MD5 value of request body and x-ots-contentmd5 in header.	The MD5 calculated based on the data in the request body is not the same as the x-ots-contentmd5 value in the request header.	No
400	OTSParameterInvalid	Missing header: ‘{HeaderName} ‘.	The request lacks a required header.	No
400	OTSParameterInvalid	Invalid date format: {Date}.	The date format is invalid.	No
400	OTSParameterInvalid	Unsupported operation: {Operation}.	The operation name in the request URL is invalid .	No

API errors

HTTPStatus	ErrorCode	ErrorMsg	Description	Retry
500	OTSInternalServerError	Internal server error.	Internal error	Yes
403	OTSQuotaExha	Too frequent	The	Yes

	usted	table operations.	CreateTable, ListTable, DescribeTable and DeleteTable operations have been executed too frequently.	
403	OTSQuotaExhausted	Number of tables exceeded the quota.	The number of tables exceeds the quota.	No
400	OTSParameterInvalid	The name of Primary Key must be unique.	The names of the Primary Key columns in the table to be created are not unique.	No
400	OTSParameterInvalid	Failed to parse the ProtoBuf message.	Deserialization of the PB data in the request body failed.	No
400	OTSParameterInvalid	Both read and write capacity unit are required to create table.	During the table creation, you must specify the **ReservedThroughput**.	No
400	OTSParameterInvalid	Neither read nor write capacity unit is set.	During the table update, you must set a reserved read or write throughput value.	No
400	OTSParameterInvalid	Invalid instance name: ‘{InstanceName} ‘.	The instance name is invalid.	No
400	OTSParameterInvalid	Invalid table name: ‘{TableName} ’.	The table name is invalid.	No
400	OTSParameterInvalid	The value of read capacity unit must be in range: [{LowerLimit}, {UpperLimit}].	The reserved read througput value must be in the specified range.	No
400	OTSParameterI	The value of	The reserved	No

	nvalid	write capacity unit must be in range: [{LowerLimit}, {UpperLimit}].	write througput value must be in the specified range.	
400	OTSParameterInvalid	Invalid column name: ‘{ColumnName} ’.	A column name is invalid.	No
400	OTSParameterInvalid	{ColumnType\is an invalid type for the Primary Key.	The Primary Key column type is invalid.	No
400	OTSParameterInvalid	{ColumnType\is an invalid type for the Primary Key in GetRange.	In the GetRange request, the Primary Key column type is invalid.	No
400	OTSParameterInvalid	{ColumnType\is an invalid type for the attribute column.	The attribute column type is invalid.	No
400	OTSParameterInvalid	The number of Primary Key columns must be in range: [1, {Limit}].	The number of Primary Key columns cannot be 0 or exceed the limit.	No
400	OTSParameterInvalid	Value of column ‘{ColumnName} ’ must be UTF8 encoding.	The value of this column must use UTF8 encoding.	No
400	OTSParameterInvalid	The length of attribute column: ‘{ColumnName} ’ exceeded the MaxLength:{MaxSize\with CurrentLength:{CellSize}.	The name of the attribute column exceeds the maximum name length limit.	No
400	OTSParameterInvalid	No row specified in the request of BatchGetRow.	In the BatchGetRow request, no rows were specified.	No
400	OTSParameterI	Duplicated	The	No

	nvalid	table name: ‘{TableName}’.	BatchGetRow or BatchWriteRow operation contains tables of the same name.	
400	OTSParameterInvalid	No row specified in table: ‘{TableName}’.	In the BatchGetRow operation, no rows were specified for a table.	No
400	OTSParameterInvalid	Duplicated Primary Key:’{PKName}‘ of getting row #{RowIndex\in table ‘{TableName}’.	In the BatchGetRow operation, some rows in some tables contain the Primary Key columns of the same name.	No
400	OTSParameterInvalid	Duplicated column name with Primary Key column: ‘{PKName}‘ while putting row #{RowIndex\in table: ‘{TableName}’.	In the BatchWriteRow operation, the PutRow operations for certain tables have rows that have an attribute column with the same name as a Primary Key column.	No
400	OTSParameterInvalid	Duplicated column name with Primary Key column: ‘{PKName}‘ while updating row #{RowIndex\in table: ‘{TableName}’.	In the BatchWriteRow operation, the UpdateRow operations for certain tables have rows that have an attribute column with the same name as a Primary Key column.	No
400	OTSParameterInvalid	“Duplicated column name: ‘{ColumnName}‘ while putting row #{Index\in table:	In the BatchWriteRow operation, the PutRow operations for certain tables have rows that	No

		‘{TableName} ’.”	contain duplicate Attribute columns.	
400	OTSParameterInvalid	“Duplicated column name: ‘{ColumnName} ‘ while updating row #{Index\in table: ‘{TableName} ’ .”	In the BatchWriteRow operation, the UpdateRow operations for certain tables have rows that contain duplicate Attribute columns.	No
400	OTSParameterInvalid	No attribute column specified to update row #{RowIndex\in table ‘{TableName} ’.	In the BatchWriteRow operation, when updating certain rows in certain tables, no attribute columns were specified.	No
400	OTSParameterInvalid	Invalid condition: {RowExistence\ while updating row #{RowIndex\in table : ‘{TableName} ’.	In the BatchWriteRow operation, when updating certain rows in certain tables, the RowExistence conditions were invalid.	No
400	OTSParameterInvalid	Duplicated Primary Key name: ‘{PKName} ‘.	Duplicate Primary Key.	No
400	OTSParameterInvalid	Invalid condition: {RowExistence\ while deleting row.	During the delete row operation, the RowExistence condition was invalid.	No
400	OTSParameterInvalid	The limit must be greater than 0.	The limit parameter must be greater than 0.	No
400	OTSParameterInvalid	Duplicated attribute column name with Primary Key column:	Some lines to be written contain an Attribute column with	No

		‘{ColumnName} ‘ while putting row.	the same name as a Primary Key column.	
400	OTSParameterInvalid	“Duplicated column name: ‘{ColumnName} ‘ while putting row.”	Some lines to be written contain duplicate attribute columns.	No
400	OTSParameterInvalid	Duplicated attribute column name with Primary Key column: ‘{ColumnName} ‘ while updating row.	Some lines to be updated contain an attribute column with the same name as a Primary Key column.	No
400	OTSParameterInvalid	No column specified while updating row.	During the update row operation, no column to be updated was specified.	No
400	OTSParameterInvalid	Duplicated column name: ‘{ColumnName} ‘ while updating row.	During the update row operation, some rows contained duplicate attribute columns.	No
400	OTSParameterInvalid	Invalid condition: {RowExistence\ while updating row.	During the update row operation, the RowExistence conditions were invalid.	No
400	OTSParameterInvalid	Optional field ‘v_string’ must be set as ColumnType is STRING.	When assigning values, the incoming column values (optional parameter) must maintain consistency with the column data type defined by the parameter. They must all be String type.	No
400	OTSParameterI	Optional field	When	No

	nvalid	'v_int' must be set as ColumnType is INTEGER.	assigning values, the incoming column values (optional parameter) must maintain consistency with the column data type defined by the parameter. They must all be Integer type.	
400	OTSParameterInvalid	Optional field 'v_bool' must be set as ColumnType is BOOLEAN.	When assigning values, the incoming column values (optional parameter) must maintain consistency with the column data type defined by the parameter. They must all be Boolean type.	No
400	OTSParameterInvalid	Optional field 'v_double' must be set as ColumnType is DOUBLE.	When assigning values, the incoming column values (optional parameter) must maintain consistency with the column data type defined by the parameter. They must all be Double type.	No
400	OTSParameterInvalid	Optional field 'v_binary' must be set as ColumnType is BINARY.	When assigning values, the incoming column values (optional parameter) must maintain consistency with the	No

			column data type defined by the parameter. They must all be Binary type.	

Table Store storage exceptions

HTTPStatus	ErrorCode	ErrorMsg	Description	Retry
503	OTSServerBusy	Server is busy.	The OTS internal server is busy.	Yes
503	OTSPartitionUnavailable	The partition is not available.	An internal server exception has caused some table partitions to be unavailable.	Yes
503	OTSTimeout	Operation timeout.	Internal OTS operation timeout.	Yes
503	OTSServerUnavailable	Server is not available.	An internal OTS server is unavailable.	Yes
409	OTSRowOperationConflict	Data is being modified by the other request.	Conflict due to multiple concurrent requests trying to write the same row.	Yes
409	OTSObjectAlreadyExist	Requested table already exists.	The table to be created by the request already exists.	No
404	OTSObjectNotExist	Requested table does not exist.	The requested table does not exist.	No
404	OTSTableNotReady	The table is not ready.	The newly created tables are not immediately available.	Yes
403	OTSTooFrequentReservedThroughputAdjustment	Capacity unit adjustment is too frequent.	The reserved read/write throughput has been adjusted too frequently.	Yes

403	OTSNotEnough CapacityUnit	Remaining capacity unit is not enough.	The remaining reserved read/write throughput is insufficient.	No
403	OTSConditionC heckFail	Condition check failed.	Pre-query condition check failed.	No
400	OTSOutOfRowS izeLimit	The total data size of columns in one row exceeded the limit.	The total data size of all columns in the row exceeds the limit.	No
400	OTSOutOfColu mnCountLimit	The number of columns in one row exceeded the limit.	The total number of columns of this row exceeds the limit.	No
400	OTSInvalidPK	Primary Key schema mismatch.	Primary Key mismatch.	No

Restricted items

Restricted Item	Restriction	Description
The number of instances saved under an Alibaba Cloud user account	Up to 10	If you need to raise the limit, please contact customer service.
The number of tables in an instance	Up to 64	If you need to raise the limit, please contact customer service.
Instance name length	3-16 bytes	Character set: [a-zA-Z0-9] and hyphens(-). The first character must be a letter and the last character cannot be a hyphen(-).
Table name length	1-255 bytes	Character set: [a-zA-Z0-9_]. The first character must be a letter or (_).
Column name length	1-255 bytes	Character set: [a-zA-Z0-9_]. The first character must be a letter or (_).
Number of columns in a Primary Key	1-4	At least 1 and no more than 4.

Size of String type Primary Key column values	Up to 1KB	A single Primary Key column’s String type column value is limited to 1 KB.
Size of String type attribute key column values	Up to 2MB	A single attribute column’s String type column value is limited to 2 MB.
Size of Binary type Primary Key column values	Up to 1KB	A single Primary Key column’s Binary type column value is limited to 1 KB.
Size of Binary type attribute key column values	Up to 2MB	A single attribute column’s Binary type column value is limited to 2 MB.
Number of attribute columns in a single row	Unlimited	A single row can contain the infinite attribute columns.
Data size of a single row	Unlimited	The total size of all column names and column value data for a single row is unlimited.
Reserved read/write throughput for a single table	1-5000	If you need to raise the limit, please contact customer service.
Number of columns in a read request’s columns_to_get parameter	0-128	Read request.
Number of UpdateTable operations for a single table	Raise: Unlimited; Lower: Unlimited	A calendar day is from 00:00:00 to 00:00:00 of the next day in UTC time, or 08:00:00 to 08:00:00 of the next day in Beijing time.
UpdateTable frequency for a single table	Once every 2 minutes	The reserved read/write throughput for a single table can be adjusted no more than once every 2 minutes.
The number of rows read by one BatchGetRow request	Up to 100	N/A
The number of rows written by one BatchWriteRow request	Up to 200	N/A
Data size of one BatchWriteRow request	Up to 4MB	N/A
Data returned by one GetRange operation	5000 rows or 4MB	The data returned by a single operation cannot exceed 5000 rows or 4 MB. If there is more data, the excessive data will be truncated.
The data size of an HTTP	Up to 5MB	N/A

Request Body		