ds.setSchema REST

Description

This method allows specifying a schema for a data type in Gigya's Data Store (DS). The schema sets field names, data types, formatting and encryption as well as client side access restrictions. Data object schemas act as meta-data, guiding Gigya how to handle the data in the specified fields.

- Depending on schema setting, other fields can be added to the storage outside the schema, Gigya will add them as is without special treatment.
- Field deletion: a field in the Data object schema is eligible for deletion if no data has been saved to it. To delete a field in the schema, you need to set it to *null* (see example below).
- Changes to schema are incremental, when setting the schema for part of the fields, the properties of the omitted fields remain unchanged.
- Unless specified otherwise, schema defined fields are only accessible from the server.

Note: For security reasons this method is not available for client side SDKs, only for server side SDKs.

Note: If you plan on integrating the DS, we highly recommend reading the Data Store Guide.
The DS is a premium platform that requires separate activation. If the DS is not part of your site package please contact your Gigya Customer Engagement Executive or email support@gigya-inc.com.

Request URL

Where **<Data_Center>** is:

- **`us1.gigya.com`** - For the US data center.
- **`eu1.gigya.com`** - For the European data center.
- **`au1.gigya.com`** - For the Australian data center.
- **`ru1.gigya.com`** - For the Russian data center.
- **`cn1.gigya-api.cn`** - For the Chinese data center.

If you are not sure of your site's data center, see Finding Your Data Center.

Parameters

Required	Name	Type	Description
	type	string	A string indicating the type of the object, equivalent to a schema name. The objective of this field is to classify objects as sharing the same schema.
	dataSchema	JSON object	A JSON object defining the schema to be set. See the format definition of the schema object below.

	format	string	Determines the format of the response. The options are: • *json* (default) • *jsonp* - If the format is jsonp then you are required to define a *callback* method (see parameter below).
	callback	string	This parameter is required only when the *format* parameter is set to *jsonp* (see above). In such cases this parameter should define the name of the callback method to be called in the response, along with the jsonp response data.
	httpStatusCodes	Boolean	The default value of this parameter is *false*, which means that the HTTP status code in Gigya's response is always 200 (OK), even if an error occurs. The error code and message is given within the response data (see below). If this parameter is set to *true*, the HTTP status code in Gigya's response would reflect an error, if one occurs.

Authorization Parameters

Each REST API request must contain identification and authorization parameters.

Some REST APIs may function without these authorization parameters, however, when that occurs, these calls are treated as **client-side** calls and all client-side rate limits will apply. In order to not reach client-side IP rate limits that may impact your implementation when using server-to-server REST calls, it is **Recommended Best Practice** to always sign the request or use a secret. A non-exhaustive list of REST APIs that this may apply to are as follows:

- accounts.login
- socialize.login
- accounts.notifyLogin
- socialize.notifyLogin
- accounts.finalizeRegistration
- accounts.linkAccounts

Please refer to the Authorization Parameters section for details.

Schema Object

Field	**Type**	**Description**
unique	Array	An array defining the fields on which you wish to impose a constraint of uniqueness. You may impose a compound uniqueness constraint based on a set of fields (see the example below). Only regular primitive data fields can be defined as unique. **Note:** Setting fields to be "unique" only works going forward, so if the fields already have non-unique values, the non-unique values will remain.

fields	JSON object	This object defines properties per data fields. The object contains field names and their corresponding list of properties (see the example below). The field name should be the full path name with dots, i.e. "*work.address*". The properties in option are: • **type** - Defines the data type of this field. The supported values are "*integer*", "*float*", "*boolean*", "*string*", "*date*", "*long*", "*text*" and "binary". When the '*type'* property is not specified the type will be deduced from the field name extension or automatically according to the content of the first item. When defining a 'binary' field type, it is not inferred from the data must be declared explicitly through the 'type' parameter. Please note that field's type cannot be changed after it contains data. • **writeAccess** - Specifies whether to allow unsigned requests to write into this field. The supported values are: ◦ "*serverOnly*" (default) - Only signed requests coming from the server are allowed. ◦ "*clientCreate*" - Unsigned requests coming from the client are allowed to write into this field, only if it wasn't set before. ◦ "*clientModify*" - Unsigned requests coming from the client are allowed to write into this field and modify existing values. • **format** - Allows assigning a regular expression that defines the format of this data field. Gigya will verify that data set in this data field matches the *format*. If the data set in this field doesn't match the *format*, Gigya will not set the data and will return an error. The best practice is to validate the data format on your client application. The *format* property accepts a string of the form: "regex('')". The regex syntax can be found in: Regular Expression Language Reference. • **languages** - An array of strings containing languages with which a text field is compatible, in addition to the default languages. A maximum of four such languages are supported. The values passed in this property are added to the existing languages in the field, and if the total number exceeds four languages, the operation fails. **Note:** The only language currently supported for this functionality is Japanese. • **encrypt** - An encryption algorithm to use for encrypting data when writing in and reading from this field. Currently the only supported value is "AES". The binary data after hashing will be stored as a Base64 string. If set, data placed in this field is stored encrypted. When you retrieve the data, Gigya decrypts it (you receive the original data). Searches on encrypted data are limited to exact matches (such as x = "ABCDE"), and not partial matches (x CONTAINS "ABC"). String search values are case sensitive. The only field types that currently support encryption are: • **string** • **text** If you set the **encrypt** property on a field that already contains data, the existing data will remain *un-encrypted*. Only new data will be encrypted. Once a field is set to **encrypt**, it is not possible to remove encryption for that field. **To delete a field** from the schema, you must ensure that data has never been saved to it (even if said data has been deleted), it has no defined **type**, and then set it to ***null*** (see "field6" in the example below). Saving data to a field automatically assigns a **type**. Setting a field's properties to null will reset the **writeAccess** property to "serverOnly" (without deleting the field), if the field contains/contained data and/or has a **type** setting.
dynamicSchema	Boolean	Specifies whether the schema is strict or dynamic. The default value of this parameter is *true* i.e. dynamic schema. In a dynamic schema you may add new fields that are not defined in the schema to the DS. The new fields are automatically added to the dynamic schema. In a strict schema you can only write into fields that are already defined in the schema.

Schema Object Example

```
{
    unique: [
       ["field1"],                         // an example of a single field
that must be unique
       ["field2", "field3" ]               // an example of a compound
unique constraint based on two fields
     ],
    fields: {
        "field1": {
            writeAccess:"clientCreate",
            format:"regex('/^[a-z0-9_-]{3,16}$/')"
        },
        "field4": {
            type:"float"
        },
        "field5": {
            type:"string",
            encrypt:"AES"
        },
        "field6": null,     // Delete field6
        "field7": {
           type:"text",
           languages:["ja"]
        }
    },
    dynamicSchema: false
}
```

Response Data

Field	Type	Description
errorCode	integer	The result code of the operation. Code '0' indicates success, any other number indicates failure. For a complete list of error codes, see the Error Codes table.
errorMessage	string	A short textual description of an error, associated with the errorCode, for logging purposes. This field will appear in the response only in case of an error.
errorDetails	string	This field will appear in the response only in case of an error and will contain the exception info, if available.
fullEventName	string	The full name of the event that triggered the response. This is an internally used parameter that is not always returned and **should not** be relied upon by your implementation.
callId	string	Unique identifier of the transaction, for debugging purposes.
time	string	The time of the response represented in ISO 8601 format, i.e., yyyy-mm-dd-Thh:MM:ss.SSSZ or
statusCode	integer	The HTTP response code of the operation. Code '200' indicates success. This property is deprecated and only returned for backward compatibility.
statusReason	string	A brief explanation of the status code. This property is deprecated and only returned for backward compatibility.

Response Example

```
{
        "statusCode": 200,
        "errorCode": 0,
        "statusReason": "OK",
        "callId": "31ba039fb8d340ceb2f43d52c89bf187",
        "time": "2015-03-22T11:42:25.943Z"
}
```