RESPONSES

Overview

When a response comes back, there are two pieces of information to consider:

HTTP Status code - this tells you at a protocol level what the status of the response is.

Response Payload - this is the body of the response. It is in a standard format that will give you the application level status, the results, and if applicable, any errors.

Response Formats

All responses are returned using JSON. JSON is a lightweight serialization language that is compatible with many different languages. JSON is syntactically correct JavaScript code, meaning you can use the Javascript eval() function to parse it.

HTTP Status Codes

Our REST API uses standard HTTP status codes to indicate success or failure of API calls. Like with most topics, Wikipedia has good resources on HTTP Statuses, but here are the basics of what we will return:

HTTP Code	**Message**	**Description**
200	OK	The request was processed and returned successfully. Nothing was changed.
201	Created	The new resource was created successfully.
400	Bad Request	Problem with the request, such as a missing, invalid or type mismatched parameter.
401	Unauthorized	The request did not have valid authorization credentials.
403	Forbidden	Private data you are not allowed to access, or you've hit a rate limit.
404	Not Found	Your URL is wrong, or the requested resource doesn't exist.
500	Server Error	We call this the "crap code" error. Basically we've got a problem on our side. If this persists please contact support. We log and review all errors but your help often

		helps us fix it quicker.
503	Service Unavailable	Our API is down. Please try again. We work hard to make this rare.

The non-20x status codes above correspond with an error of one type or another. Below we detail the error responses.

Response Payload

Each API response is wrapped in a standard payload that contains the results of the API call, and depending of the type of API call will result in one of two payloads. The lookup response payload is an object containing the data and optionally metadata such as timestamp or relationsId while the search response payload contains a list of data objects and some additional useful information and metadata, such as the total count of records and the type of the result object.

Lookup response payload:

```
{
   "metadata": {
   },
   "data": {
      "lseId": 2756,
      "name":"Georgia Power Co",
      "code":"7140",
      "websiteHome":"http://www.georgiapower.com/"
   }
}
```

The above shows a successful response back from the call. If the requested object does not exist an error 404 is returned.

Search response payload:

```
{
   "status": "success",
   "count": 2,
   "type": "LoadServingEntity",
   "data":[ {
      "lseId": 2756,
      "name":"Georgia Power Co",
      "code":"7140",
      "websiteHome":"http://www.georgiapower.com/"
   }, {
      "lseId":1,
      "name":"City of Augusta",
      "code":"1000",
      "websiteHome":null
   }]
}
```

The above shows a successful response back from the call. In this case the result type is LoadServingEntity, and there are two results.

Name	Type	Description
status	String	Possible values are: success and error.
count	Integer	Total count of records that match your request. Because of pagination, this is not necessarily the number of records in the response.
type	String	Resource (sometimes called the Object) type.
data	Object[]	This contains the actual results, i.e. the object or objects that you requested.

Large response payload

Since a search response payload could hit an upper resource limit we sometimes need to split the response in chunks. Since every search response payload will contain the count field indicating the total number of data objects the client will be able to handle receiving the payload in chunks. The way the server handles this is to check for a certain parameter in the request which contains the number of the first data object to send and the the server will send the next chunk of data objects starting from this number. Data objects are numbered from 1..n. An example:

First request: http://foo.bar/getService

First response:

```
{
    "status": "success",
    "count": 2,
    "type": "LoadServingEntity",
    "data":[{
        "lseId": 2756,
        "name":"Georgia Power Co",
        "code":"7140",
        "websiteHome":"http://www.georgiapower.com/"
    }]
}
```

Second request: http://foo.bar/getService?offset=2

Second response:

```
{
   "status": "success",
   "count": 2,
   "type": "LoadServingEntity",
   "data":[{
      "lseId":1,
      "name":"City of Augusta",
      "code":"1000",
      "websiteHome":null
   }]
}
```

Response Payload Changes

From time to time, we may add to the set of fields that is returned from a particular endpoint in the API. In general, these new fields will only appear if you specify that the API should return the extended set of fields. However, API clients should always be prepared to handle (i.e. ignore) any new object properties that they do not recognize.

Errors

Errors are communicated to the client in the payload of the response.

Errors in the Response Payload

The standard response payload can contain errors. This will give more finely grained details about specific what caused the error or errors, which should allow you to respond appropriately. This might include displaying validation messages to your users, or changing the request at runtime.

```
{
   "status":"error",
   "count":2,
   "type":"Error",
   “relationId”:”42795780-3056-11e9-bba9-005056b9e8b4”,
   “timestamp”:”2019-02-14T17:20:07.866+01:00”,
   "data":[{
      "code":"NotNull",
      "message":"An appKey must be supplied",
      "objectName":"requestSignature",
      "propertyName":"appKey"
   }, {
      "code":"NotNull",
      "message":"An appId must be supplied",
      "objectName":"requestSignature",
      "propertyName":"appId"
   }]
}
```

Error Type Definition

The type Error has the following data definition:

Name	**Type**	**Description**
code	String	Error code. Unique string identifying the type of error. See the table below for a list of possible error codes. (Always returned).
message	String	Localized user readable message describing the error. (Conditionally returned)
objectName	String	Identifies the type of object that the error is related to. This is typically a resource type that is part of the response. For example, the Tariff object. (Conditionally returned)
propertyName	String	Identifies the property of the object that the error is related to. Primarily for binding and validation errors, but sometimes used for business logic errors too. For example, the tariffId or lseId. (Conditionally returned)
relationId	String	An id uniquely identifying the Mule message
timestamp	Datetime	The date and time the error occurred.

List of Error Codes

The code in the Error can be any of the following:

Name	**Description**
ObjectNotFound	Typically this is when the item id you passed in was not found. For example, passing in a non-existent id.
NotNull	A required object was not passed in on the request. Check the objectName and propertyName for more information.

InvalidArgument	An argument passed in did not meet validation requirements. This may or may not be related to the request parameters passed in.
InvalidState	The current state of the object in question prevents the requested action from being performed.
TypeMismatch	An argument was not of the expected type. This may or may not be related to the request parameters passed in.
MissingId	An ID is required to process the request, but none was supplied.
DataIntegrityViolationException	The data sent in was not valid. You could see this error when trying to overwrite a value that cannot be overwritten.
InvalidFileFormat	The file format sent in was invalid.
InvalidError	The supplied value is not valid.
ObjectCannotBeSaved	There was an error saving the object.
InsufficientPermissions	The supplied credentials do not have permission to access this resource.
SystemError	Unexpected exception.
UniquenessViolationError	You can only have one object with this ID. Usually seen when trying to create a new object with an Id that already exists.
InvalidDateRange	The date range is not valid. Usually caused by a fromDateTime being after a toDateTime.