Creating and Modifying Items

POST: Create a New Item document

POSTing to [] will create a new item, however there are two basic requirements that must be met to create the item. First, POST request must include a valid parentId, and second, it must have a title. Here we will send the simplest JSON formatted request that we can and take a look at its output.

> ⚠ Testing creating items and making sure your parameters are correct and are being set is easily done using a browser based REST client such as REST Client for Firefox. See Using RESTClient for Firefox for more information.

To create a single record, send POST request to URL: https://www.sciencebase.gov/catalog/item/

Simple Request

```
{"parentId":"4f28888de4b050c1ade5f38e","title":"Test Item Title"}
```

Response (formatted sbJSON)


```
{
  "link":
  {
    "rel":"self",
    "url":"http://www.sciencebase.gov/catalog/item/4f2ac20ee4b020113d57d88a"
  },
  "id":"4f2ac20ee4b020113d57d88a",
  "title":"Test Item Title",
  "dateCreated":"2012-02-02 10:04:14 MST",
  "lastUpdated":"2012-02-02 10:04:14 MST",
  "hasChildren":false,
  "parentId":"4f28888de4b050c1ade5f38e",
  "permissions":
  {
    "read":
    {
      "acl":["PUBLIC"],
      "inherited":true,
      "inheritsFromId":"4f287da8e4b050c1adc28acc"
    },
    "write":
    {
      "acl":null,
      "inherited":true,
      "inheritsFromId":"4f287da8e4b050c1adc28acc"
    }
  }
}
```


In the response we can see all of the default values that are given to an item, the new item's id and that the title and parentId that we used to create the item are set correctly. You may note that this item is inheriting its permissions from an item id that is not the parentId. This is normal as the parent of this item is inheriting its permissions from an item above it, these values are passed down (and dynamic).

More Advanced POST Requests

Adding more fields to your POST requests is as simple as understanding the type of request you are sending. In JSON it means adding more key/value pairs that match the key/value pairs you are trying to set. For example, a "subtitle" is a String just like the item title so you would put:

```
"subtitle":"<subtitle text>"
```

More advanced types are a little trickier, but an easy way to figure out the correct format is to do a GET on an object that already has these properties and see what type of object the "value" is. AlternateTitles, for example, are an array of Strings, so you need to create the array of Strings and then set that as the "value" of the key/value pair you are adding. In REST Client this looks simple:

```
"alternateTitles":["<alt title 1>", "<alt title 2>"]
```

In Groovy however it is easier to create the array of Strings (as an array or a List) and then add that as the value in your Map or JSON object.

Some values are more complicated than just arrays of Strings. Tags, for example, are represented as an Array of Maps. In REST Client this can be sent as:

```
"tags":[{"name":"<tag1>"},{"name":"<tag2>"}]
```

It is important to test and make sure you understand the structures you are adding onto, thus using REST Client

PUT: Update an Existing Item document

A PUT is very similar to a POST except that you need to use the full URL for the item, eg. http://www.sciencebase.gov/catalog/item/4f2ac20ee4b020113d57d88a instead of http://www.sciencebase.gov/catalog/item/ that we used in the POST. Most fields will just overwrite any fields that you put new data in, this is how you are able to clear data from a field, set it as a blank String, i.e. removing all the tags would be:

```
{"tags":""}
```

Using Natural Identifiers in PUT requests

It is also possible to update a document using one of its natural identifiers (stored in the `identifiers` field of the item). The only restriction is that the identifier you are using must be unique within the set of items that your user may update. Otherwise, the REST api will return a 409 (Conflict) HTTP response for more than one matching item.

There are two ways to specify the item identifier. In both cases, you *must* not include a ScienceBase identifier in the URL. You will simply use the http://www.sciencebase.gov/catalog/item/ base url.

- You may specify the identifier as query parameters. For example, http://www.sciencebase.gov/catalog/item/?scheme=CSC&type=id&key=108593.
- Alternately, if you omit the base ScienceBase identifier and the identifier query parameters, the system will use the first item identifier specified in the JSON data included in the PUT request. For example, with the following item data PUT to http://www.sciencebase.gov/catalog/item/, the system would look for a record with the **first** identifier. It will completely ignore any subsequent records.

  ```
  {
    title: "My Updated Title",
    identifiers: [{scheme: "CSC", type: "id", key: "108593"},{scheme: "DND", type: "foo", key: "11111"}]
  }
  ```

UPSERT: Create or update

An upsert is the combination of "update" and "insert" or POST and PUT. This functionality is useful when you have a natural identifier for an item, but don't know if it already exists in your user's writeable area of ScienceBase. The upsert API lets you submit a full item document, along with the known natural identifiers. ScienceBase will look for an existing item that matches the first item identifier included in the document you submit. If it finds it, that item will be updated. Otherwise, a new item is created.

NOTE: you must specify a "parentId" in the item document or ScienceBase will be unable to create the item.

Upsert requests should be POST-ed to http://www.sciencebase.gov/catalog/item/upsert. You may optionally specify the natural identifiers as query parameters rather than specifying them in the POST body (e.g. http://www.sciencebase.gov/catalog/item/upsert?scheme=CSC&type=id&key=108593). If the natural identifiers are not specified as query parameters, the **first** identifier in the item's data. All other identifiers would be ignored.

```
{
  parentId: "4f3d3bf1ccf2089542bc925c",
  title: "My Updated Title",
  identifiers: [{scheme: "CSC", type: "id", key: "108593"}]
}
```

This example would search for an item matching the CSC identifier 108593 and update or create the item as appropriate.

Exception behaviors:

- If a ScienceBase item id is specified in the POST-ed JSON, that item will be updated if it is writable. Otherwise, 403 or 404 will be returned as appropriate. An item will never be created in this instance.
- If more than one *writable* item matches the identifier provided, no action is taken and a 409 error is returned.
- If no *writable* item matches the identifier, but one or more *readable* items do match the identifier, no action is taken a 403 error is returned.
- If exactly one *writable* item matches the identifier provided, regardless of the number of *readable* items that match, the *writable* item will be updated.

POST: Multiple Items

Creating multiple items with one REST call is very similar to creating a single item. The first difference is that since we are creating *items* we will use the *items* controller: https://www.sciencebase.gov/catalog/items The second difference is that although sbJSON of each individual item is exactly the same you treat each item as an entry in an array:

```
[{title: "Test item 1", parentId: 5033f109e4b068b9cdc547f8}, {title: "Test item 2", parentId:
5033f109e4b068b9cdc547f8}]
```

The returned sbJSON will also be in an array and will contain all of the same information that a single call would, for each item:

```
[
    {
        "link":
        {
            "rel": "self",
            "url": "https://beta.sciencebase.gov/catalog/item/504695ede4b0e82aebe2cb0f"
        },
        "id": "504695ede4b0e82aebe2cb0f",
        "title": "Test item 1",
        "dateCreated": "2012-09-04T17:59:41-06:00",
        "lastUpdated": "2012-09-04T17:59:41-06:00",
        "hasChildren": false,
        "parentId": "5033f109e4b068b9cdc547f8",
        "permissions":
        {
            "read":
            {
                "acl":
                [
                    "PUBLIC"
                ],
                "inherited": true,
                "inheritsFromId": "5033f17ae4b068b9cdc547fb"
            },
            "write":
            {
                "acl":
                [
                    "USER:sbdocumentationtesting"
                ],
                "inherited": true,
                "inheritsFromId": "5033f17ae4b068b9cdc547fb"
            }
        }
    },
    {
        "link":
        {
            "rel": "self",
            "url": "https://beta.sciencebase.gov/catalog/item/504695ede4b0e82aebe2cb11"
        },
        "id": "504695ede4b0e82aebe2cb11",
        "title": "Test item 2",
        "dateCreated": "2012-09-04T17:59:41-06:00",
        "lastUpdated": "2012-09-04T17:59:41-06:00",
        "hasChildren": false,
        "parentId": "5033f109e4b068b9cdc547f8",
        "permissions":
        {
            "read":
            {
                "acl":
                [
                    "PUBLIC"
                ],
                "inherited": true,
                "inheritsFromId": "5033f17ae4b068b9cdc547fb"
            },
            "write":
            {
                "acl":
                [
                    "USER:sbdocumentationtesting"
                ],
                "inherited": true,
                "inheritsFromId": "5033f17ae4b068b9cdc547fb"
            }
        }
    }
]
```

PUT: Multiple Items

PUT's to update multiple items act on the same principals that POST's do. First, the call must be made to the items controller: https://www.sciencebase.gov/catalog/items. For the input sbJSON since we are not calling the item by id in our URL we must put the id of the item you wish to update in the sbJSON for each item:

```
[{title: "Test item 1.2", id: 504695ede4b0e82aebe2cb0f}, {title: "Test item 2.2", id: 504695ede4b0e82aebe2cb11}]
```

The returned sbJSON will be of the same type as the sbJSON returned by a mutliple items POST.

Using Natural Identifiers in PUT requests

Note that natural identifiers may also be used for using PUT with multiple items. Read more about using natural identifiers with PUT in the single-item PUT section above.

```
[{title: "Test item 1.2", identifiers:[{scheme: "CSC", type: "id", key: "12345"}]}, {title: "Test item 2.2",
id: 504695ede4b0e82aebe2cb11}]
```

UPSERT: Multiple Items

Upserts for multiple items work very similarly to the upserts for single items, but may be batched. The POST should be made to http://www.sciencebase.gov/catalog/items/upsert and will contain an array of JSON objects. The semantics of each individual upsert operation is identical to those described in the section above for single upserts.

```
[{title: "Test item 1.2", identifiers:[{scheme: "CSC", type: "id", key: "12345"}]}, {title: "Test item 2.2",
identifiers:[{scheme: "CSC", type: "id", key: "999999"}}]
```

Asynchronous POSTs, PUTs, UPSERTs, and DELETEs

If you are doing calls to create, update or delete a large number of items it may be advantageous to not have to wait for all the items to be modified in SB to get your return back. In this case you can use the async parameter which will first return a status page with a 202 accepted header and, when the operations are complete in SB, return the sbJSON of the actions you submitted.

Making an asynchronous call is the same as making a normal call, but with the async parameter set to true. For example if you are doing a POST of multiple items the URL that you are POSTing to will be https://www.sciencebase.gov/catalog/items?async=true.

The response will be a JSON object with the keys requestId and status and a Status Code Header with the value "202 Accepted". The requestId will be the id that you can use to access the requests status at on the async controller. Using this requestId you can continue to check the status of your call at https://www.sciencebase.gov/catalog/async/<requestId>. The status will first return a more direct path to the asynchronous call's page and then the status of the call once it starts to be processed.

Example return:

```
{
    "requestId": "504777dae4b010bea98f8231",
    "status": "/catalog/async/504777dae4b010bea98f8231"
}
```