Overview

Try Custom Code in less than 5 minutes

Setup

Datastore

Request Data

Queries

Relationships

Authentication

Geolocation

Push Notifications

Logging

External HTTP Calls

Custom Headers and Response

Caching

External Dependencies

Building Custom Code

Deploying to Production

Testing Locally

Best Practices

Restrictions

Custom Code Developer Guide

Overview

Custom Code is Java, Scala, or Clojure code you write that runs on the StackMob server.

You deploy it to the servers by either uploading your JAR or linking your GitHub repository with StackMob.

Important note regarding Scala: StackMob has officially deprecated Scala 2.9.x and older. Developers using Scala to write Custom Code methods are advised to upgrade to Scala 2.10.x as soon as possible.

If you wrote a Custom Code method called `hello_world`, then StackMob creates an API endpoint for you at `https://api.stackmob.com/hello_world` so that it's callable from the REST API and StackMob Client SDKs. The server will execute the server side custom code and return a response with a body that you define.

Example of mobile SDKs calling Custom Code:

iOS SDK | Android SDK | JS SDK | REST API

iOS calling Custom Code `hello_world`

```
SMCustomCodeRequest *request = [[SMCustomCodeRequest alloc]
  initGetRequestWithMethod:@"hello_world"];

[[[SMClient defaultClient] dataStore] performCustomCodeRequest:reque
  onSuccess:^(NSURLRequest *request,
          NSHTTPURLResponse *response,
          id responseBody) {
        NSLog(@"Success: %@",responseBody);
  } onFailure:^(NSURLRequest *request,
          NSHTTPURLResponse *response,
          NSError *error,
          id responseBody){
        NSLog(@"Failure: %@",error);
}];
```

Custom Code allows you to easily communicate between client and server.

In each section of this guide you may see colored boxes which are meant to highlight important information:

Gold boxes call out warnings, gotchas, and information we don't want you to miss.

Blue boxes contain links to sections in the full API reference, as well as full working projects for you to download and in-depth tutorials for you to read through.

To cover a universal audience, this developer guide will be covered in Java, though the fundamentals apply to each supported language of Scala and Clojure as well.

Code examples of the topics covered in this Developer Guide can be found in our GitHub repositories below.

Resources

- StackMob Custom Code Java examples on GitHub
- StackMob Custom Code Scala and Clojure examples on GitHub

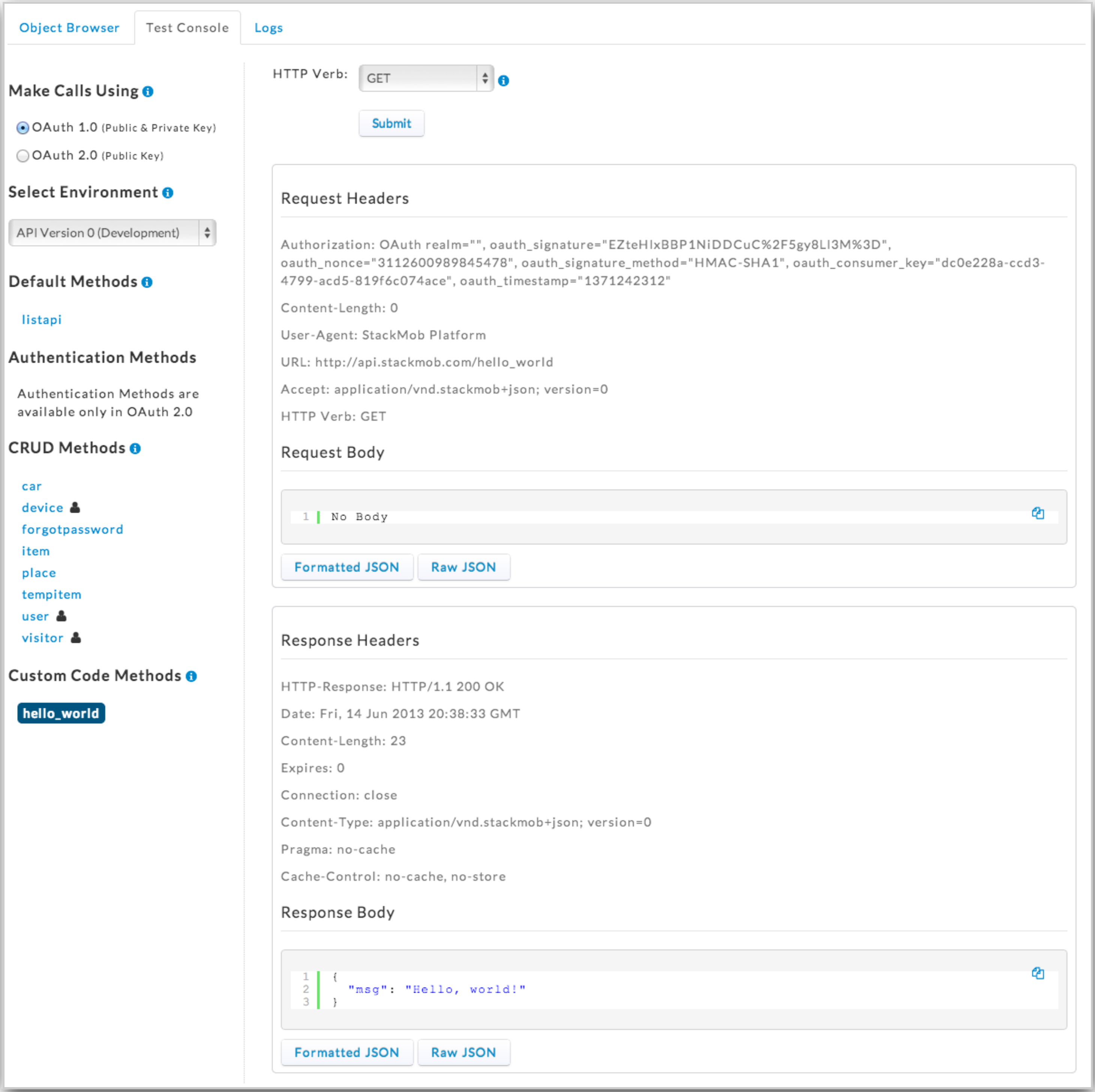
Try Custom Code in less than 5 minutes

We've provided a ready-to-upload JAR so you can try Custom Code immediately. It will introduce a `hello_world` function on the server side that, when called, returns JSON `{ msg: "Hello, World!"}`.

1. Add the Custom Code module.
2. Download the Hello World example JAR
3. Upload it through the Dashboard.

That's it! You can now try Custom Code in the Dashboard Console, a UI to help you make REST API requests against your StackMob API (and hence your custom code).

Below is the result of selecting the `hello_world` endpoint and submitting a `GET` request:



Authentication Methods are available only in OAuth 2.0

HTTP Verb: GET

Request Headers

```
Authorization: OAuth realm="", oauth_signature="EZteHlxBBP1NiDDCuC%2F5gy8LI3M%3D", oauth_nonce="3112600989845478", oauth_signature_method="HMAC-SHA1", oauth_consumer_key="dc0e228a-ccd3-4799-acd5-819f6c074ace", oauth_timestamp="1371242312"
Content-Length: 0
User-Agent: StackMob Platform
URL: http://api.stackmob.com/hello_world
Accept: application/vnd.stackmob+json; version=0
HTTP Verb: GET
```

Request Body

```
No Body
```

Response Headers

```
HTTP-Response: HTTP/1.1 200 OK
Date: Fri, 14 Jun 2013 20:38:33 GMT
Content-Length: 23
Expires: 0
Connection: close
Content-Type: application/vnd.stackmob+json; version=0
Pragma: no-cache
Cache-Control: no-cache, no-store
```

Response Body

```
{
  "msg": "Hello, world!"
}
```

Notice that you're making a REST API call to `hello_world`. Custom Code methods are converted into REST API endpoints and are accessible from anywhere you can make an HTTP request with headers.

You just uploaded and ran your first Custom Code method!

Custom Code methods are converted into REST API endpoints and are accessible from anywhere you can make an HTTP request with headers.

Resources

- View the example Custom Code Java Starter project

Setup

Let's get a Custom Code project started so that you can start adding your own methods (API endpoints). We've provided an easy way to get started below. It's a zip file with all the necessary files needed to build a project. (It's a Maven project, so be sure to install Maven.)

Download the Custom Code Java Starter Template zip and unzip it.

If you want to build your JAR right away, simply run `mvn package` in the root folder (where `pom.xml` is).

We've included `src/main/java/com/stackmob/customcode/HelloWorld.java` which you can remove. But it's there as an example, and we'll use it here in this Developer Guide. (Note it's also referred to in `EntryPointExtender.java`)

Use GitHub? You can skip Maven and JARs. Fork the StackMob Custom Code Java Starter repository and link your Custom Code repository with StackMob. StackMob will automatically build your project for you.

Class Template

A method is represented by a class that extends the interface `CustomCodeMethod`, of which there are three methods to implement:

- `getMethodName`
- `getParams`
- `execute`

Let's look at the basic class, represented by `HelloWorld.java`.

```
public class HelloWorld implements CustomCodeMethod {

  @Override
  public String getMethodName() {
    return "hello_world"; //no dashes or spaces allowed
  }

  @Override
```

```java
  public List<String> getParams() { return new ArrayList<String>(); }

  @Override
  public ResponseToProcess execute(ProcessedAPIRequest request, SDKServic
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("msg", "Hello, world!");
    return new ResponseToProcess(HttpURLConnection.HTTP_OK, map);
  }

}
```

This method will return JSON when called:

```
{ msg: "Hello, world!" }
```

`getMethodName` defines your API endpoint: *https://api.stackmob.com/*`hello_world`.

`getParams` whitelists the parameters you would pass into the method from the URL. (It does not fetch parameters from the POST body because StackMob custom code doesn't support www-url-form-encoded requests at this time)

`execute` runs when the REST API point for `https://api.stackmob.com/hello_world` is hit. The hashmap that is returned will translate to a JSON object in the response.

⚠ **getMethodName**: we do not support dashes (`-`) in method names, but we do support underscores (`_`).

API References

- CustomCodeMethod

Exposing the Method

For StackMob to be aware of this method, you should add an entry for `HelloWorld` in `src/main/java/com/stackmob/customcode/EntryPointExtender.java`. **This is a mandatory step.**

```java
public class EntryPointExtender extends JarEntryObject {
  @Override
  public List<CustomCodeMethod> methods() {
    List<CustomCodeMethod> list = new ArrayList<CustomCodeMethod>();
    list.add(new HelloWorld());
    return list;
  }

}
```

⚠ Methods **must be entered** in `EntryPointExtender` in order for StackMob to discover them.

Datastore

Custom Code can access your datastore via the Custom Code SDK (included in your project via Maven's `pom.xml` automatically).

Custom Code bypasses access control permissions since it runs in your secure environment, so even if you have CRUD permissions set to `Not Allowed` in your schema permissions, Custom Code can access it.

You'll be accessing the datastore via `DataService`.

Examples

- There are several fully working Custom Code Datastore examples available.

Create

Let's create an object in the datastore from custom code.

```
public ResponseToProcess execute(ProcessedAPIRequest request,
        SDKServiceProvider serviceProvider) {
    DataService ds = serviceProvider.getDataService();

    HashMap<String, Object> car = new HashMap<String, Object>();

    car.put("model", new SMString(model)); //string
    car.put("make", new SMString(make)); //string
    car.put("year", new SMInt(Long.parseLong(year))); //int

    try {
      // This is how you create an object in the `car` schema
      ds.createObject("car", new SMObject(car));
    } catch (InvalidSchemaException ise) {
    } catch (DatastoreException dse) {}
}
```

API References

- DataService#createObject

Examples

- Create Object Example

Read

Let's read an object from the datastore.

```
@Override
public ResponseToProcess execute(ProcessedAPIRequest request,
        SDKServiceProvider serviceProvider) {

    DataService ds = serviceProvider.getDataService();
    List<SMCondition> query = new ArrayList<SMCondition>();
    Map<String, List<SMObject>> results = new HashMap<String, List<SMObj

    try {
        //Get the car with ID "12345"
        query.add(new SMEquals("car_id", new SMString("12345")));

        // Read objects from the car schema
        results.put("results", ds.readObjects("car", query))

    } catch (InvalidSchemaException ise) {
    } catch (DatastoreException dse) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, results);
}
```

API References

- DataService#readObjects

Examples

- Read Object Example

Update

Let's update an object.

```
public ResponseToProcess execute(ProcessedAPIRequest request,
        SDKServiceProvider serviceProvider) {

    DataService ds = serviceProvider.getDataService();
    List<SMUpdate> update = new ArrayList<SMUpdate>();
    update.add(new SMSet("year", new SMInt(Long.parseLong(year))));

    Map<String, SMValue> results = new HashMap<String, SMValue>();

    try {
        //Update the year of the car "12345"
        results.put("results", ds.updateObject("car", new SMString("12345
    } catch (InvalidSchemaException ise) {
    } catch (DatastoreException dse) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, results);
}
```

API References

- DataService#updateObject

Examples

- Update Object Example

Delete

Let's delete an object.

```
public ResponseToProcess execute(ProcessedAPIRequest request,
        SDKServiceProvider serviceProvider) {

    DataService ds = serviceProvider.getDataService();

    try {
      ds.deleteObject("car", new SMString(carID));
    } catch (InvalidSchemaException ise) {
    } catch (DatastoreException dse) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- DataService#deleteObject

Examples

- Delete Object Example

Request Data

Whether making queries or performing CRUD operations, you'll likely want to take user input to do so.

You can:

- fetch parameters out of the URL for GET and DELETE requests.
- fetch data/JSON out of the request body for PUT and POST requests.

Fetching Parameters

Let's get the parameters out of the request URL. To start out, let's first make a GET request from the client SDKs with a few parameters.

iOS SDK | Android SDK | JS SDK

```
SMCustomCodeRequest *request = [[SMCustomCodeRequest alloc]
    initGetRequestWithMethod:@"hello_world"];

[request addQueryStringParameterWhere:@"name" equals:@"joe"];
[request addQueryStringParameterWhere:@"age" equals:[NSNumber numbe

[[[SMClient defaultClient] dataStore] performCustomCodeRequest:reque
  onSuccess:^(NSURLRequest *request,
              NSHTTPURLResponse *response,
              id responseBody) {
        NSLog(@"Success: %@",responseBody);
  } onFailure:^(NSURLRequest *request,
               NSHTTPURLResponse *response,
               NSError *error,
               id responseBody){
        NSLog(@"Failure: %@",error);
}];
```

These result in `https://api.stackmob.com/hello_world?name=joe&age=10`.

Now let's get the parameters out of the request in custom code.

```
public ResponseToProcess execute(ProcessedAPIRequest request,
    SDKServiceProvider serviceProvider) {

  String name = request.getParams().get("name"); //this will be a String
  String age = request.getParams().get("age"); //this will be a String

  return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

Fetching JSON Body

Perhaps you're sending up JSON. Let's do that with the client SDKs.

iOS SDK | Android SDK | JS SDK

```
SMCustomCodeRequest *request = [[SMCustomCodeRequest alloc]
    initGetRequestWithMethod:@"hello_world"];

NSDictionary *body = [NSDictionary dictionaryWithObjectsAndKeys:@"j
NSErrtor *error = nil;
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:body
```

```
  options:NSJSONWritingPrettyPrinted error:&error];
if (error) {
  // Handle error
}

request.requestBody = [[NSString alloc] initWithData:jsonData encon

[[[SMClient defaultClient] dataStore] performCustomCodeRequest:reque
  onSuccess:^(NSURLRequest *request,
              NSHTTPURLResponse *response,
              id responseBody) {
        NSLog(@"Success: %@",responseBody);
  } onFailure:^(NSURLRequest *request,
               NSHTTPURLResponse *response,
               NSError *error,
               id responseBody){
        NSLog(@"Failure: %@",error);
}];
```

These send up a request of:

```
URL:
https://api.stackmob.com/hello_world

Body:

{name:'joe',age:10}
```

Let's see how we can pull that out of the JSON body.

```
import org.json.JSONException;
import org.json.JSONObject;

...

public ResponseToProcess execute(ProcessedAPIRequest request,
    SDKServiceProvider serviceProvider) {

  try {
    JSONObject jsonObj = new JSONObject(request.getBody());
    if (!jsonObj.isNull("places")) {
      places = Arrays.asList(jsonObj.getString("places").split(","));
    }
  } catch (JSONException e) {
    logger.error("Doh!  Problem parsing the JSON.", e);
  }

  return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

StackMob returns `request.getBody()` as a plain String, so you'll need to mold it into the format you want - in this case JSON.

Examples

- Extracting JSON out of the request body example

Queries

Let's make some queries against the datastore.

Fetching Multiple Results

At the heart of fetching objects is the `DataService`. The `DataService` is retrieved from the request which is passed into the `execute` method. To read several objects, you'd use the `readObjects` method.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
    DataService ds = serviceProvider.getDataService();
    ...
    results = ds.readObjects("car", query, 0, filters);
}
```

Let's look at `readObjects` more closely. The method is overloaded, so let's take a look at the expanded one.

```
List<SMObject> readObjects(String schema,
                           List<SMCondition> conditions,
                           int expandDepth,
                           ResultFilters resultFilters)
                           throws InvalidSchemaException,
                                  DatastoreException
```

We'll cover each of the parameters in more detail below, but here's an overview:

- `schema` - the name of schema you're querying
- `conditions` - the list of conditions which comprise the query (less than, greater than)
- `expandDepth` - the depth to which a query should be expanded for relationships (return full objects X levels deep) - we'll cover this later in the Relationships section. For now, you can assume this value can be 0.
- `resultFilters` - the options to be used when filtering the resultset (pagination, ordering)

API References

- DataService#readObjects

Examples

- Read All Objects Example

Equality

Let's look for users with the birthyear "2000".

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    List<SMCondition> query = new ArrayList<SMCondition>();

    String year = "2000"

    DataService ds = serviceProvider.getDataService();
    List<SMObject> results;

    try {
      query.add(new SMEquals("birthyear", new SMInt(Long.parseLong(year)
      results = ds.readObjects("user", query, 0, filters);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- DataService#readObjects
- SMEquals

Comparison

You can query for greater than/less than. Let's get all users with a `birthyear` field greater or equal to the year 2000.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
    List<SMCondition> query = new ArrayList<SMCondition>();

    String year = "2000"

    DataService ds = serviceProvider.getDataService();
    List<SMObject> results;

    try {
      // We only want years greater than or equal to the user input
      query.add(new SMGreaterOrEqual("birthyear", new SMInt(Long.parseLon
      results = ds.readObjects("user", query, 0, filters);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- DataService#readObjects
- SMGreater
- SMGreaterOrEqual
- SMLess
- SMLessOrEqual
- SMNotEqual
- there are more in the docs!

Examples

- Comparison Query Example

Or Queries

You can include `or` statements in your query logic. Let's try to find `car`s whose `make` is `Ferrari` `or` whose `year` is newer than `2010`.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
  List<SMCondition> orArguments = new ArrayList<SMCondition>();
  orArguments.add(new SMGreaterOrEqual("year", new SMInt(2000L)));
  orArguments.add(new SMEquals("make", new SMString(make)));

  SMOr orStatement = new SMOr(orArguments);

  List<SMCondition> query = new ArrayList<SMCondition>();
  DataService ds = serviceProvider.getDataService();
  List<SMObject> results;

  try {
    // add the OR statement (containing the conditions from orArguments)
    query.add(orStatement);
    results = ds.readObjects("car", query);
  } catch (InvalidSchemaException ise) {}
    catch (DatastoreException dse) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- SMOr

Examples

- Or Query Example

Array Queries

You can query for objects to see if an array or relationship contains a value.

Say a `user` had an array of `friends`, and you want to get those who are friends with `john` and `jane`.

```java
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    List<SMCondition> query = new ArrayList<SMCondition>();
    List<SMValue> values = new ArrayList<SMValue>();
    values.add(new SMString("john"));
    values.add(new SMString("jane"));
    query.add(new SMIn("friends", values));

    ...

    DataService ds = serviceProvider.getDataService();
    List<SMObject> results;

    try {
      results = ds.readObjects("user", query, 0, filters);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

This works for both array and relationship fields.

API References

- DataService#readObjects
- SMIn
- SMValue
- SMInt
- SMLong
- SMString
- SMBoolean

Pagination Queries

Fetch a few results at a time. Let's return items 5 through 9.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    List<SMCondition> query = new ArrayList<SMCondition>();

    ResultFilters filters = new ResultFilters(0, 9, null, null);

    DataService ds = serviceProvider.getDataService();
    List<SMObject> results;

    try {
      results = ds.readObjects("user", query, 0, filters);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- ResultFilters
- DataService#readObjects

Examples

- Pagination Example

Ordering

You can sort results and even provide tie breakers. We are going to primarily sort by year (oldest to most recent) and then by createddate from newest to oldest.

Pass `SMOrdering` specifiers into `ResultFilters`.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    List<SMOrdering> orderings = Arrays.asList(
            new SMOrdering("year", OrderingDirection.ASCENDING),
            new SMOrdering("createddate", OrderingDirection.DESCENDING))
    ResultFilters filters = new ResultFilters(0, -1, orderings, null); /

    DataService ds = serviceProvider.getDataService();
    List<SMObject> results;

    try {
      query.add(new SMGreaterOrEqual("year", new SMInt(Long.parseLong(ye
      results = ds.readObjects("user", query, 0, filters);

    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- SMOrdering
- DataService#readObjects

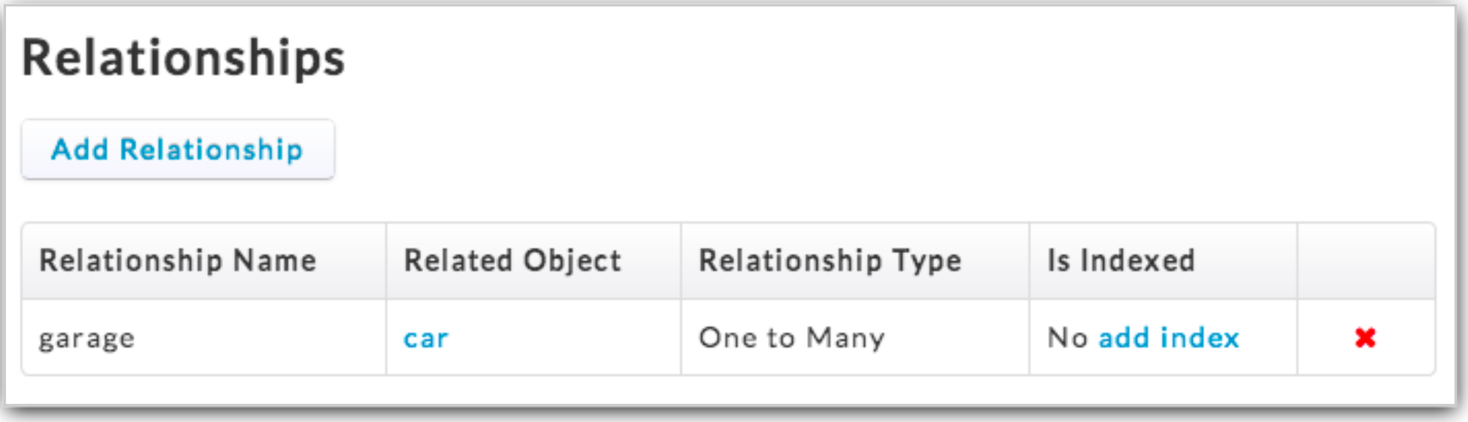
Examples

- Equality Query with Ordering Example

Relationships

Just as in our SDKs, you can manipulate relationships from the Custom Code SDK.

Let's assume we have a `user` schema, and we've related the user to the `car` schema. Let's add cars to the user on the `garage` field, which we've created as a Relationship field.


Relationships

Add Relationship

Relationship Name	Related Object	Relationship Type	Is Indexed	
garage	car	One to Many	No add index	✖

Adding related objects

Existing Child Object Instances

If the `car` already exists in the datastore, and you want to relate it with a `user`. You just need to relate the two existing objects together with the primary keys.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    DataService ds = serviceProvider.getDataService();

    List<SMValue> relatedObjects = new ArrayList<SMValue>();
    relatedObjects.add(new SMString("Camry")); //primary keys of car obj
    relatedObjects.add(new SMString("Accord"));
    try {
      SMObject result = ds.addRelatedObjects("user", new SMString("john"
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

No new objects are created in the datastores. We're just linking existing objects with each other.

API References

- DataService#addRelatedObjects

Examples

- Add Existing Object to Parent Example

New Child Object Instances

If the cars *don't* exist in the `car` schema yet, we can create them and relate them to the `user` in one call. Let's give a `user` two new `cars`. The following will create the two `car` objects in the respective `car` schema and relate them to the user at the same time.

```
public ResponseToProcess execute(ProcessedAPIRequest request,
      SDKServiceProvider serviceProvider) {
    // These are some example cars that will be created
    Map<String, SMValue> carValues1 = new HashMap<String, SMValue>();
    carValues1.put("make", new SMString("Audi"));
    carValues1.put("model", new SMString("R8"));
    carValues1.put("year", new SMInt(2005L));

    Map<String, SMValue> carValues2 = new HashMap<String, SMValue>();
    carValues2.put("make", new SMString("Audi"));
    carValues2.put("model", new SMString("spyder"));
    carValues2.put("year", new SMInt(2005L));

    SMObject car1 = new SMObject(carValues1);
    SMObject car2 = new SMObject(carValues2);

    List<SMObject> cars = new ArrayList<SMObject>();
    cars.add(car1);
    cars.add(car2);

    DataService ds = serviceProvider.getDataService();

    try {
      BulkResult result = ds.createRelatedObjects(
        "user", new SMString("john"), "garage", cars);

      feedback.put(owner + " now owns", cars);

    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

The user should now look something like:

```
{
    username: 'john',
    garage: ["151gd", "1351dg5"],
    ...
}
```

API References

- DataService#createRelatedObjects

Examples

- Create and Add New Object to Parent Example

Fetching related objects

To retrieve related objects from the datastore, you can simply call `readObjects`. Normally the related objects are represented by the primary keys:

```
{
    username: 'john',
    garage: ['Camry', 'Accord']
}
```

But you can get **expanded related objects** by passing an expand depth of 1.

```
{
    username: 'john',
    garage: [{
        car_id: 'Camry',
        maker: 'Toyota',
        ...
    }, {
        car_id: 'Accord',
        maker: 'Honda',
        ...
    }]
}
```

Here's the code.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    DataService ds = serviceProvider.getDataService();
    int expandDepth = 1;
    try {
        List<SMObject> results = ds.readObjects("user",
            new ArrayList<SMCondition>(), expandDepth);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- DataService#readObjects

Examples

- Expanded Fetch Example

Authentication

The StackMob client SDKs support OAuth 2.0 login. When they make a request to custom code, custom code is aware of the logged in user.

```java
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
    if (request.getLoggedInUser() != null) loggedInAction();
    else notLoggedInAction();
    ...
    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- ProcessedAPIRequest#getLoggedInUser

Geolocation

Persisting Geopoints

To manipulate geolocations in Custom Code, we'll just prepare the `lat` and `long` values as `SMDouble` instances. We'll then pass them as a `SMSet` into our CRUD operations.

Let's update `john`'s home with a new geolocation value.

```java
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService

    DataService ds = serviceProvider.getDataService();

    Map<String, SMValue> geoPoint = new HashMap<String, SMValue>();
    geoPoint.put("lat", new SMDouble(37.772201));
    geoPoint.put("lon", new SMDouble(-122.406326));

    List<SMUpdate> update = new ArrayList<SMUpdate>();
    update.add(new SMSet("home", new SMObject(geoPoint)));

    try {
      SMObject result = ds.updateObject("user", new SMString("john"), up
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- SMDouble
- SMSet
- DataService#updateObject

Examples

- Saving GeoPoints

Querying Geopoints

Let's query for several users who live within ~60 miles of us.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP

    SMNear near = new SMNear(              // Near-condition results will al
            "position",                    // name of GeoField in schema
            new SMDouble(37.77207),        // latitude
            new SMDouble(-122.40621),      // longitude
            new SMDouble(.0025));          // radius - (62.25 mi) can be nul

    SMWithinBox withinBox = new SMWithinBox(  // Whereas withinbox result
            "position",
            new SMDouble(37.8),
            new SMDouble(-122.47),         // Top Left coords
            new SMDouble(37.7),
            new SMDouble(-122.3));         // Bottom Right coords

    DataService ds = serviceProvider.getDataService();

    List<SMCondition> query = new ArrayList<SMCondition>();
    query.add(near);
    query.add(withinBox);


    try {
      List<SMObject> results = ds.readObjects("user", query);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
  }
```

StackMob geolocation distances are in radians. .0025 radians is around 62.25 miles.

API References

- DataService#readObjects
- SMCondition
- SMWithin
- SMWithinBox
- SMNear

Examples

- Querying GeoPoints

Push Notifications

Registering Devices

Let's pass a `device_token` to the method and register it. Here, we'll register it to a particular username so that in the future, we can send push messages to StackMob usernames rather than device tokens. That'll make things a bit easier.

Because StackMob supports both Apple and Google Push notifications, you need to specify the type as well.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    //Specify the token type
    TokenType deviceTokenType = TokenType.iOS;

    //Can also be TokenType.AndroidGCM or TokenType.Android (the latter

    String deviceToken = request.getParams().get("device_token"); //devi
    TokenAndType token = new TokenAndType(deviceToken, deviceTokenType);

    try {
      PushService service = serviceProvider.getPushService();
      service.registerTokenForUser(username, token);
    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

You've now registered the device token so that StackMob can send messages to it.

API References

- PushService#registerTokenForUser

Examples

- Register Push Example

Broadcast Messages

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
    Map<String, String> payload = new HashMap<String, String>();

    try {
      PushService ps = serviceProvider.getPushService();
      // Add data to your push payload
      payload.put("key1", "value1");
      payload.put("sound", "someSound.mp3");
      payload.put("alert", "Push Alert!");

      ps.broadcastPush(payload);

    } catch (Exception e) {}

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

API References

- PushService#broadcastPush

Examples

- Broadcast Push Notification Example

Direct Messages

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    Map<String, String> payload = new HashMap<String, String>();

    try {
      PushService ps = serviceProvider.getPushService();

      // Add data to your payload
      payload.put("badge", "1");
      payload.put("key1", "some data");

```

```
        // Send the payload to the specified user
        ps.sendPushToUsers(Arrays.asList("john"),payload);

      } catch (Exception e) {}

      return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
  }
```

API References

- PushService#sendPushToUsers

Examples

- Send Direct Push Notifications to Users Example

Logging

You can write logs that you can view at your Dashboard Logs.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKServiceP
    LoggerService logger = serviceProvider.getLoggerService(Logging.class

    logger.info("This is an INFO log");
    logger.error("This is ");
    logger.warn("This is ");
    logger.debug("This is ");

    return new ResponseToProcess(HttpURLConnection.HTTP_OK, ...);
}
```

Examples

- Logging Example

External HTTP Calls

You can make calls to external APIs from custom code, but for security purposes, they must go through our `HTTPService` provider.

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    // The service you're going to be using
    String url = "http://www.httpbin.org/get";

    // Formulate request headers
    Header accept = new Header("Accept-Charset", "utf-8");
    Header content = new Header("Content-Type", "application/x-www-form-

    Set<Header> set = new HashSet();
    set.add(accept);
    set.add(content);

    try {
      HttpService http = serviceProvider.getHttpService();

      /* In this Example we are going to be making a GET request
       * but PUT/POST/DELETE requests are also possible.
       */
      GetRequest req = new GetRequest(url,set);
      HttpResponse resp = http.get(req);

      responseCode = resp.getCode();
      responseBody = resp.getBody();

    } catch (Exception e) {}

    return new ResponseToProcess(responseCode, ...);
}
```

API References

- HttpService
- GetRequest
- PostRequest
- PutRequest
- DeleteRequest

Examples

- Making an External HTTP Call

Custom Headers and Response

You can send up custom headers in your custom code API calls.

You can also send back custom response headers and response bodies in various formats (non-JSON).

Caching

The SDK includes functionality in `CachingService` to store key/value data in a fast, distributed cache. If your custom code does expensive computation or long running I/O, you should consider caching the results to make your code more efficient.

For example, if your code does expensive datastore queries, and the queries don't need to be fresh for each request, you could increase its performance like this (exception handling omitted for clarity):

```
public ResponseToProcess execute(ProcessedAPIRequest request, SDKService
    CachingService cache = provider.getCachingService();
    String result = cache.get("expensive-query");
    if(result == null) {
        result = executeExpensiveQuery();
        cache.set("expensive-query", result, 1000); //store the result f
    }

    return new ResponseToProcess(responseCode, ...);
}
```

A few more notes:

- The above pattern works if the datastore query can be up to 1 second stale. We recommend that you cache as much as possible and query only the parts that need to be fresh.
- We recommend holding temporary data in `CachingService` rather than memory wherever possible
- All of your app's cache data is namespaced, so it won't be overwritten by another app
- `CachingService` limits the size of each key and value, and rate limits the number of `get` and `set` calls your app can make.
- `CachingService` `get`s and `set`s are almost always faster than `DatastoreService` operations

API References

- CachingService

External Dependencies

If you're working with external libraries, you can include them with Maven or include the JAR.

Maven

Maven helps you build your projects by also organizing your dependencies. Many developers upload their JARs to Maven's central repository, allowing you, the developer, to simply define what resource you need in Maven's xml. Maven will pull it in for you automatically to help build your project.

Below we include Google's `json-simple` library to help with JSON manipulation

```xml
<dependencies>
  ...
  <dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
  </dependency>
  ...
</dependencies>
```

Resources

- Example pom.xml

Building Custom Code

You can build your Custom Code with `Maven`, or if your project is in a GitHub repository, you can use our UI to connect StackMob with your GitHub repo and we'll build it automatically for you.

Maven | GitHub | sbt

Using Maven to build your Custom Code JAR

For Java, StackMob recommends using Maven.

Our projects have `Maven`'s `pom.xml` file included, so you just have to go to the Terminal and navigate to where your `pom.xml` file is and run:

```
mvn package
```

This will build the JAR which you can upload to StackMob. Your JAR can be found at:

```
your-project-folder-name/target/stackmob-customcode-build-0.1.0-SNAPS
```

Upon uploading, StackMob will deploy your JAR to your development environment.

All Maven files and folders are already configured for you in the Starter project above.

Resources

- A Maven Custom Code Scala example

Deploying to Production

To get your Custom Code to the Production environment, just deploy your API. Deploying your API will roll out both schemas and custom code.

Testing Locally

You can run Custom Code locally on a mock API server StackMob provides. This local custom code dev environment houses your JAR and proxies API calls. You would treat this running server as your test API server and point your requests at it.

Read about setting up and running your local custom code dev environment.

Best Practices

Be sure to read about Custom Code Best Practices. It'll keep your code running quickly and will help keep your code scalable!

Restrictions

The Custom Code environment has some restrictions so as to ensure a secure environment.

- can not create new threads
- can not read/write files
- can not read/write to socket (unless using StackMob's HTTPService)
- can not read/write properties
- can not use reflection
- maximum execution time is 25 seconds