Handshakez Salesforce API

# Introduction

A key product offering of Handshakez (HSZ) is its Salesforce.com (SFDC) integration. This integration includes custom Salesforce objects, a managed Salesforce package, and an API communication layer between HSZ and SFDC. This document will focus on the API layer.

# Revision History

|  |  |  |
| --- | --- | --- |
| Date | Author | Comment |
| 9/12/2013 | Alan Arvesen | Document rooms. |
| 9/10/2103 | Alan Arvesen | Document opportunities and user activity. |
| 9/9/2013 | Alan Arvesen | Improved documentation of dailyLogin resource. Basic REST usage. |
| 9/6/2013 | Alan Arvesen | Initial revision. |

# General Description

The API is based on a REST paradigm. For the purposes of this document, this means:

* The API is accessed via URLs
* Each URL roughly represent a single “resource” in HSZ
* These resources respond to various HTTP verbs, such as GET, POST, and PUT

Beyond REST, there are a few common implementation details:

* Access is secured via a custom hashing scheme
* REST infrastructure is provided by the “tastypie” project (<http://django-tastypie.readthedocs.org/en/latest/>)
* Rudimentary online documentation provided by “swagger” (<https://pypi.python.org/pypi/django-tastypie-swagger/0.1.1>, http://swagger.wordnik.com/)

# Basic REST Usage

The REST URL is composed of several pieces. For example:

/api/salesforce/v1/room/

In order, the path elements represent:

|  |  |
| --- | --- |
| Path Element | Meaning |
| api | Root of API access |
| salesforce | API module (salesforce or mobile) |
| v1 | API version (as of this 9/1/2013, all are v1) |
| room | Desired resource |

To request a different resource, the final path element would be changed to the desired resource (e.g. from room to userActivity).

When requesting the URL, the resource will respond to the HTTP “verb” or type of request. These requests are generally “read” (GET), “update” (PUT), or “create” (POST). This follows standard REST usage. Note that not every resource supports every REST verb. In particular, no resources currently implements DELETE.

Many resources respond to an additional esource identifier. For instance, to access room number 1, you could append a 1 to the above URL:

/api/salesforce/v1/room/1/

Some of the URLs respond to additional parameters. These are detailed in the per-resource sections below. All of the URLs require additional authentication parameters passed in the query string. These are detailed in the authentication section. Finally, the Swagger module will provide rudimentary API information. This information can be accessed via the “doc” version of the REST URL. For example:

/api/salesforce/doc/

TODO: discuss JSON submit/response?

TODO: HOW-TO setup the connector and required DB fields for users? Or is this a separate document?

# Authentication and Authorization

Authentication in the API layer is achieved using a combination of a secret key, a public token, and email address, and a timestamp.

The secret key and token are generated for each user when their account is created. Locally, these are stored in the table tastypie\_apikey as key and key\_token, respectively.

Note

API access is restricted to HTTPS connections.

TODO: is this true? Hah. Not on staging!

In order to connect to the API, the client first constructs the desired REST URL (e.g. /salesforce/v1/room). To the end of this URL the following parameters are appended:

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Notes |
| email\_id | Email address | Email address of user |
| timestamp | Integer | Seconds since epoch. Timestamp of address request (TODO: UTC?) |
| api\_token | Hex string | Value from tastypie\_apikey.key\_token. Not a secret. |
| api\_key\_hash | Hex string | The API request hashed with tastypie\_apikey.key. See below for hashing details. |

There are three different facets to authentication and authorization. These are time sensitive requests, hashing with the secret key, checking user privileges.

## Time sensitivity

All API calls include a parameter called timestamp. This value indicates the number of seconds since the Unix epoch. The server requires that this timestamp be within three minutes of the current time. To prevent an attacker from naively changing the timestamp and resubmitting, the timestamp is included in the hashed message.

## Hashing

To avoid sending the secret key as plain text, a hashing mechanism is used. This is accomplished using the HMAC message digest routine as described in RFC 2104. This routine takes three inputs: a secret key, a message body, and a hashing algorithm. The API key is used as the secret. SHA1 is used as the hashing algorithm. The message that is hashed is slightly complicated.

* We begin by examining the parameters in the query string. That is, we look for everything after the question mark in the query string. So, for example, in /salesforce/v1/room/?email\_id=foo@bar.com&flavor=chocolate&topping=nuts, we would treat email\_id,flavor and topping as parameters.
* Next, we ignore the parameters email\_id and api\_key\_hash. So in the example above, we would only care about flavor and topping.
* Next, we reassemble these parameters into a list of name value pairs preceded by a question mark. So, in our example, we would have the value “?flavor=chocolate&topping=nuts”
* Next, we add the timestamp
* Finally, we add the secret key

TODO: should this actually hash the path as well? Since the path is important in the REST query

From the client side, the hash is added to the request as the api\_token parameter. From the server side, the computed hash value is compared to the api\_token parameter to ensure that the request has not been tampered with.

## User Privileges

After a user has been authenticated via the hashing scheme, we check to see if he is authorized to use the API.

Users are identified by the api\_token and email\_id parameters. The API user identified by the API token must be the admin of an organization. The email user identified by the email parameter does not need to be the same user; instead, we look up the user using the field salesforce\_email in the local database. If we cannot find an email user, then we assume that the email user is the API user.

TODO: Is the use case here that an entire organization can use a single API key, but we can still distinguish its members based off of email? Is there a malicious case where you can have bogus email credentials but a valid api token?

One the email user has been determined, we check his privileges. The current implementation requires that the user either have the has\_subscription or the show\_unlimited\_rooms properties. “Has subscription” further devolves to stripe\_customer\_id. If either of these properties true then the user is authorized and API access proceeds.

# Currently Available Resources

## Room

The room resource provides information about Room objects in HSZ. This resource provides several different facilities. It will return information about one or many rooms. It will update a room with an opportunity id from sales force. It will also create invites.

A main consumer of this resource is the room connector run via SFDC. This is an hourly job that synchronizes the room temperature and other information between HSZ and SFDC.

TODO: this is used in the room connector, correct? Should this be described in a separate document?

Authentication: scheme described above

Authorization: a custom authorization scheme!

If a POST: then either it’s a request for all rooms, which we let anyone do (TODO: really?), or it’s a request for a single room.

If a GET, then it has to be a room request, else it’s an error

If requesting a single room, then that room must either include the requesting user as any role on the room (owner, buyer, or seller), or the room must be owned by someone in the requesting user’s organization.

Additional query parameters: TODO

Verbs: GET, PUT, POST

Fields: 'id', 'title', 'temperature', 'opportunity\_id', 'account\_id', 'unique\_room\_visits', 'new\_unique\_room\_visits',

'post\_count', 'comment\_count', 'hifive\_count', 'emails\_count', 'new\_post\_count', 'new\_comment\_count',

'new\_hifive\_count', 'new\_emails\_count', 'seller\_company', 'buyer\_company', 'hashkey', 'total\_room\_visits',

'new\_total\_room\_visits'

### Sample Request

/api/salesforce/v1/room

### Authentication

Scheme described above.

### Authorization

Custom authorization scheme. First, we try to find if there is a room id in the GET or POST. If it’s a POST and there is no room\_id, then this is a request for all rooms, and authorization is granted. If it’s a GET and there is no room requested, then it’s an error, and authorization is denied (TODO: is this correct? This should just return no data?)

If a room\_id is found, then we check to see if the room requested is either owned by the requesting user’s organization or if the requesting user has a UserRoom associated with the user. If either is true, then authorization is granted (TODO: on a post, only the very first room is tested. Isn’t this a security hole, where you can request the first valid room then request additional unowned rooms? Or am I misreading the [0] bit in the python?)

### Additional query parameters

None.

### Verbs

GET, POST, PUT

### Fields Returned

|  |  |  |
| --- | --- | --- |
| Field | Meaning | Example |
| allow\_unlimited\_rooms  company  date\_joined  email  is\_active  is\_invited  job\_title  last\_login  last\_name  linkedin\_id  organisation\_id  username | User and profile DB fields. TODO: no first name? | - |
| login\_count | Total count of logins from useractivitystats | 5 |
| has\_opted\_out\_of\_email | Inverse of user.receieve\_email | 1 |
| num\_buyer\_rooms  num\_seller\_rooms  num\_owner\_rooms | Count of rooms from userrooms where role is 1 (buyer), 2(seller) or 3 (owner) |  |
| resource\_uri | REST URI |  |

### Sample return JSON

{

"meta":

{

"limit": 20,

"next": null,

"offset": 0,

"previous": null,

"total\_count": 1

},

"objects": [

{

"allow\_unlimited\_rooms": true,

"company": "Josh",

"date\_joined": "2013-09-10T16:02:55.607308", "email": "a2@dynamicfixture.com", "has\_opted\_out\_of\_email": false,

"id": "2",

"is\_active": true,

"is\_invited": false,

"job\_title": "",

"last\_login": "2013-09-10T16:02:55.607186", "last\_name": "2",

"linkedin\_id": null,

"login\_count": 25,

"num\_buyer\_rooms": 2,

"num\_owner\_rooms": 3,

"num\_seller\_rooms": 1,

"organisation\_id": null,

"resource\_uri":

"/api/salesforce/v1/user\_activity/2/", "username": "2"

}]

}

## User Activity

The user\_activity resource provides a rolled up view of a user’s information. This information includes number of rooms by role, date joined, last login, first and last name, and other information. Unlike daily\_login, this resource provides a snapshot of data across all time rather the data segmented by date.

The dataset returned is dependent on the requesting user. If the requesting user is a member of the HSZ organization, then information about all users is returned. Otherwise, only information about users relevant to the requesting user’s organization is returned. This dataset includes all the users of the appropriate organization plus the users of rooms owned by the organization.

TODO: the “is HSZ” organization is denoted by org id 1. This should be an actual data element. Additionally, we should rationalize all that “unlimited rooms” stuff so that we have a clearer authorization model. Isn’t there django support for this?

TODO: what is the big sql doing at the top of this class?

### Sample Request

/api/salesforce/v1/dailyLogin

### Authentication

Scheme described above.

### Authorization

Default authorization scheme. (TODO: which is…?)

### Additional query parameters

None.

### Verbs

GET, POST (TODO: post?)

### Fields Returned

|  |  |  |
| --- | --- | --- |
| Field | Meaning | Example |
| allow\_unlimited\_rooms  company  date\_joined  email  is\_active  is\_invited  job\_title  last\_login  last\_name  linkedin\_id  organisation\_id  username | User and profile DB fields. TODO: no first name? | - |
| login\_count | Total count of logins from useractivitystats | 5 |
| has\_opted\_out\_of\_email | Inverse of user.receieve\_email | 1 |
| num\_buyer\_rooms  num\_seller\_rooms  num\_owner\_rooms | Count of rooms from userrooms where role is 1 (buyer), 2(seller) or 3 (owner) |  |
| resource\_uri | REST URI |  |

### Sample return JSON

{

"meta":

{

"limit": 20,

"next": null,

"offset": 0,

"previous": null,

"total\_count": 1

},

"objects": [

{

"allow\_unlimited\_rooms": true,

"company": "Josh",

"date\_joined": "2013-09-10T16:02:55.607308", "email": "a2@dynamicfixture.com", "has\_opted\_out\_of\_email": false,

"id": "2",

"is\_active": true,

"is\_invited": false,

"job\_title": "",

"last\_login": "2013-09-10T16:02:55.607186", "last\_name": "2",

"linkedin\_id": null,

"login\_count": 25,

"num\_buyer\_rooms": 2,

"num\_owner\_rooms": 3,

"num\_seller\_rooms": 1,

"organisation\_id": null,

"resource\_uri":

"/api/salesforce/v1/user\_activity/2/", "username": "2"

}]

}

## Opportunity

The opportunity resource provides a way to tie SFDC Opportunity to HSZ Rooms. When an Opportunity is updated in SFDC, the new information can be pushed to HSZ. This will create a new rooms\_roomopportunity record for each push.

### Sample Request

/api/salesforce/v1/user\_activity/

### Authentication

Scheme described above.

### Authorization

Default authorization scheme.

### Additional query parameters

None

### Verbs

GET, POST

### POST JSON Data Structure

The POST request expects to receive a JSON data structure like the following. Most of these fields correspond to the fields of the same name in Salesforce. In the sample below, ‘XXX’ would be replaced with an actual opportunity id.

{

"stage\_name": "stage",

"create\_date": "2013-08-01", "sfdc\_opportunity\_id": "XXX",

"probability": 10,

"expected\_revenue": 5588,

"close\_date": "2013-11-10T03:07:43", "forecast\_category\_name": "cat name", "account\_name": "test account",

"room": "/api/salesforce/v1/room/1/", "opportunity\_type": "new business",

"amount": 200

}

The “room” field is potentially confusing. This refers to an object that already exists in HSZ. It can be specified as REST URI as indicated. TODO: tastypie allows a PK and a dictionary as well – document these.

### Fields Returned

The opportunity resource returns a JSON object. This object has all of the same fields as the original request with the addition of a resource\_uri and id fields. These identify the opportunity which was just created. In theory, the URI could be resubmitted via a GET to return the opportunity just created. In practice, GETs are ignored.

|  |  |  |
| --- | --- | --- |
| Field | Meaning | Example |
| Id | ID of new opportunity | 2 |
| resource\_uri | URI of new opportunity | /api/salesforce/v1/opportunity/2/ |

### Sample return JSON

{

"account\_name": "test account",

"amount": "200",

"close\_date": "2013-11-10T03:07:43", "create\_date": "2013-08-01", "expected\_revenue": "5588", "forecast\_category\_name": "cat name",

"id": "2",

"opportunity\_type": "new business", "probability": "10",

"resource\_uri": "/api/salesforce/v1/opportunity/2/",

"room": "/api/salesforce/v1/room/1/", "sfdc\_opportunity\_id": "XXX",

"stage\_name": "stage"

}

## Daily Login

The dailyLogin resource provides login counts. This information includes total logins per user by date.

### Sample Request

/api/salesforce/v1/dailyLogin

### Authentication

Scheme described above.

### Authorization

Default authorization scheme.

### Additional query parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Param | Req’d | Meaning | Example |
| date | No | Limit data to specified date | date=2013-09-0 |
| user | No | Limit data to specified user id | user=123 |

### Verbs

GET only

### Fields Returned

|  |  |  |
| --- | --- | --- |
| Field | Meaning | Example |
| Date | Date of login info | 2013-09-0 |
| User | User id of login info | 123 |
| login\_count | Count of logins for this user on this day | 5 |
| total\_login | Total times ever this user has logged on (TODO: this probably isn’t useful) | 100 |

### Sample return JSON

{

"meta": {

"limit": 20,

"next": null,

"offset": 0,

"previous": null, "total\_count": 3

},

"objects": [

{

"date": "2013-01-01",

"login\_count": 1,

"resource\_uri": "/api/salesforce/v1/dailyLogin/1/", "total\_login": 3,

"user": 1

} ,

{

"date": "2013-02-01",

"login\_count": 2,

"resource\_uri": "/api/salesforce/v1/dailyLogin/2/", "total\_login": 3,

"user": 1

} ,

{

"date": "2013-01-01",

"login\_count": 10,

"resource\_uri": "/api/salesforce/v1/dailyLogin/3/", "total\_login": 10,

"user": 2

}

]

}

# Future Enhancements

Daily login – is this really daily activity?

Custom auth is a bit of a mess. Should be split into authentication and authorization pieces. Hash should incorporate the path if that is what we care about.