HOTSOCKET

API Technical Specification

Version 1.1.4.4
Author(s): Gerrie Swart
Thursday 23rd July, 2015

Contents

1 Overview 4

2 Design Goal 5

3 Formatting Stuff 5

4 Web Service 6

5 Protocol 6

6 Encoding 6

7 Resources 7

8 Test Resources 7

9 Login 8
- 9.1 Resource . . . 8
- 9.2 POST Fields . . . 8
- 9.3 Response Fields . . . 8
- 9.4 Note about the token format . . . 8
- 9.5 Login Flowchart . . . 9
- 9.6 Sample Login Responses . . . 10
  - 9.6.1 Login failure, incorrect credentials . . . 10
  - 9.6.2 Login Success . . . 10

10 Recharge 11
- 10.1 Resource . . . 11
- 10.2 POST Fields . . . 11
- 10.3 Response Fields . . . 12
- 10.4 Recharge Flowchart . . . 13
- 10.5 Example Responses . . . 14
  - 10.5.1 Successful submission . . . 14
  - 10.5.2 Invalid Token . . . 14
  - 10.5.3 Expired Token . . . 14
  - 10.5.4 Malformed MSISDN . . . 15
  - 10.5.5 Non-numeric characters in MSISDN . . . 15
  - 10.5.6 Unknown Product . . . 15
  - 10.5.7 Unknown Network . . . 15
  - 10.5.8 Network - Product - Denom combo not valid . . . 16
  - 10.5.9 Duplicate reference . . . 16
  - 10.5.10 Non-numeric reference . . . 16
- 10.6 Recipient MSISDN format . . . 16
- 10.7 Error IDs . . . 17

11 Status Lookup 18
11.1 Resource . . . 18
11.2 POST Fields . . . 18
11.3 Response Fields . . . 18
11.4 Recharge Status Codes . . . 19
11.4.1 Messages returned . . . 19
11.5 Example Responses . . . 20
11.5.1 Recharge Successful . . . 20

12 Retrieving Lists of Recharges 21
12.1 Resource . . . 21
12.2 POST Fields . . . 21
12.3 Response Fields . . . 22
12.4 Example Response . . . 22

13 Balance Lookup 23
13.1 Resource . . . 23
13.2 POST Fields . . . 23
13.3 Response Fields . . . 23
13.4 Example Responses . . . 23

14 Code Snippets 25
14.1 PHP, basic login, print results (using cURL) . . . 25
14.2 Python login, print results (httplib) . . . 25
14.3 Ruby login, print results . . . 26
14.4 Java, login, print results . . . 26

15 Contact Us 28
15.1 Website . . . 28
15.2 Email . . . 28
15.3 Johannesburg Address . . . 28
15.4 Cape Town Address . . . 28
15.5 Telephone . . . 28

16 Appendix A - Networks and Products 29
16.1 Network codes . . . 29
16.2 Product Code and Denomination combinations . . . 30
16.2.1 Cell C . . . 30
16.2.2 MTN . . . 30
16.2.3 Telkom Mobile . . . 31
16.2.4 Vodacom . . . 31

Listings

1 Login failure, JSON . . . 10
2 Login failure, XML . . . 10
3 Login success, JSON . . . 10
4 Login success, XML . . . 10
5 Successful recharge submission, XML . . . 14
6 Successful recharge submission, JSON . . . 14
7 Invalid Token, XML . . . 14
8 Invalid Token, JSON . . . 14
9 Expired Token, XML . . . 14
10 Expired Token, JSON . . . 15
11 Malformed MSISDN, XML . . . 15
12 Non-numeric MSISDN, XML . . . 15
13 Non-numeric MSISDN, JSON . . . 15
14 Unknown Product, JSON . . . 15
15 Unknown Network, JSON . . . 15
16 Network, Product, Denom combo invalid, XML . . . 16
17 Network, Product, Denom combo invalid, JSON . . . 16
18 Duplicate reference, XML . . . 16
19 Duplicate reference, JSON . . . 16
20 Reference is not numeric, JSON . . . 16
21 Status returning Successful Recharge, XML . . . 20
22 Status returning Successful Recharge, JSON . . . 20
23 Statement, XML . . . 22
24 Balance, XML . . . 23
25 Balance, JSON . . . 24
26 Basic Login, PHP . . . 25
27 Basic Login, Python . . . 25
28 Basic Login, Ruby . . . 26
29 Basic Login, Java . . . 26

1. Overview

This is a technical specification for the HOTSOCKET interface. Currently, HOTSOCKET allows you to effect PIN-less* recharges on Cell C, MTN, Telkom, and Vodacom.

The interface works on a very simple model, where requests are HTTP-POSTed to a URL, responses are in plain XML or JSON, and security is via TLS.

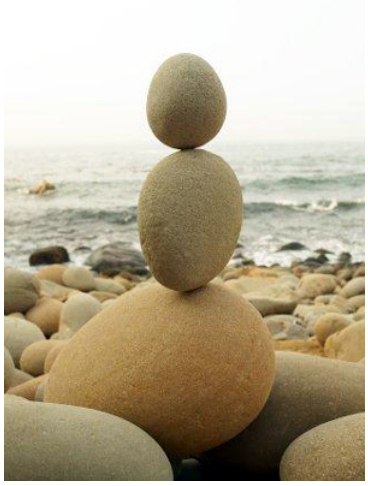
HOTSOCKET PIN-less recharges work as follows (a nearly sequential guide to issuing a PIN-less recharge):

1. You log in
2. You POST a recharge
3. You check the status of a recharge
4. You check your purse balance

* **PIN-less recharges, sometimes called direct topup**: the airtime and bundles just appear on the recipient's SIM, without anyone having to type in PIN numbers from vouchers, etc.

2. Design Goal

"Simple is better than complicated"



All design and engineering decisions are taken with this in mind. As Henry Mencken said, For every problem there is a solution which is simple, clean and wrong.

What we meant to say is "Everything should be made as simple as possible, but no simpler".[†]

3. Formatting Stuff

Formatting of this document is straightforward, but the following styles are useful to recognise:

- `keyword`: keyword or constant, the exact text to pass to HOTSOCKET , or sometimes just stuff that will typically be typed somewhere.
- `$ fortune -o`: command that can be typed into your OS (without the dollar sign).
- external link: hyperlink to an external location.
- internal link: a link internal to the document, e.g. footnotes / table of contents.
- **oh dear**: typically, rather important info.

We try to make the code readable in this document - this means we format XML and JSON where you might receive markup without whitespace, and also that certain things will contain linebreaks for formatting (e.g. `recharge_status_code` in the `/status` response) where the actual code won't.

[†] Attributed to A. Einstein by some, but seems to be a modern version of his much clumsier sentence. See quoteinvestigator.com.

4. Web Service

This isn't one. SOAP won't clean this thing, so it wasn't used.

This is an unusual thing, something that you simply POST to (RFC 2616 Section 9.5) and it will reply with a simple Plain Old XML (POX) or JavaScript Object Notation (JSON) answer. We tried to make this as simple as is sensibly possible.

It will probably help to think of this as misunderstood RESTful design. The overriding principle in choosing this setup ~~is to give people the ability to avoid nasty libraries and languages~~ is a love of simplicity and a pragmatic view of the IT ecosystem - we want diverse languages and frameworks plugging into our service.

☛ **JSON users** - we use org.json's XML to JSON code to turn the POX into JSON - it works well, but it seems to have its own ideas about when to handle the status codes we return as strings and when to handle them as numbers. Always comparing our returned statuses against numbers should do the trick.

5. Protocol

This service will run over HTTPS. Plain (unencrypted) HTTP is currently supported, though we are working toward only allowing HTTPS in future. Everything will be done by POSTing to various URLs (we refer to these URLs as *resources* in this document), and by then receiving documents of type TEXT/XML or APPLICATION/JSON.

6. Encoding

ISO 8859-1 (Latin 1) will be used throughout. You can read up on it here: http://en.wikipedia.org/wiki/ISO_8859-1, if that is the sort of thing you like to do.

If you are using Java, you can tell the VM about this decision using the `file.encoding` attribute, for example:

```
$ java -Dfile.encoding=ISO-8859-1 MyMainClass
```

'There Ain't No Such Thing As Plain Text.'
-Joel Spolsky
(http://www.joelonsoftware.com/articles/Unicode.html)

---

7. Resources

There are five resources. You will need to POST to each resource, supplying data for the fields mentioned, and you will then receive a JSON or XML response (no prologue / XML declaration, just an ISO 8859-1 encoded response). The root element is `response` in all cases. For the love of chocolate, please do not try to GET any resource, HOTSOCKET does not support it. We have some POST code snippets further in this document.

1. You will log in here:
   http://api.hotsocket.co.za:8080/login/
2. Submit recharges here:
   http://api.hotsocket.co.za:8080/recharge/
3. Retrieve the status of a single recharge here:
   http://api.hotsocket.co.za:8080/status/
   (checking the status is important because you get billed once the recharge is successful)
4. Obtain statements here:
   http://api.hotsocket.co.za:8080/statement/
5. Retrieve your balance here:
   http://api.hotsocket.co.za:8080/balance/

8. Test Resources

The test resources follow the same pattern as the real ones, but prefix `/test` before the resource name, i.e.:

1. You will log in here:
   http://api.hotsocket.co.za:8080/test/login/
2. Submit recharges here:
   http://api.hotsocket.co.za:8080/test/recharge/

And so forth.

When you first sign up for a HOTSOCKET account we will furnish you with a separate username and login for testing, and this will only work on the test resources.
Recharges submitted to the test resources will never trigger real network recharges, and the statuses you get back for tests will be chosen randomly, and will include error statuses. Balances retrieved from the test resources are also random (spread between -R 249.99 and R 1499.99).

9. Login

You log in by sending your username and password, as supplied by Flickswitch, to the /login resource. If your credentials check out, you will receive a token that you use during the duration of your session.

This token is time limited; it will expire after two hours. If it expires you will receive a response with the status code 0889. A successful login returns a status code of 0000 plus a token, while wrong credentials returns 5010.

9.1. Resource

http://api.hotsocket.co.za:8080/login/
or
http://api.hotsocket.co.za:8080/test/login/ for testing.

9.2. POST Fields

Field Name	Description	Example
username	Your username. Max 25 chars, alphanumeric.	LeonTheProfessional
password	Your password.	Tr0gd0r
as_json	**Optional**. Send true if you want the results as JSON and not XML.	true

9.3. Response Fields

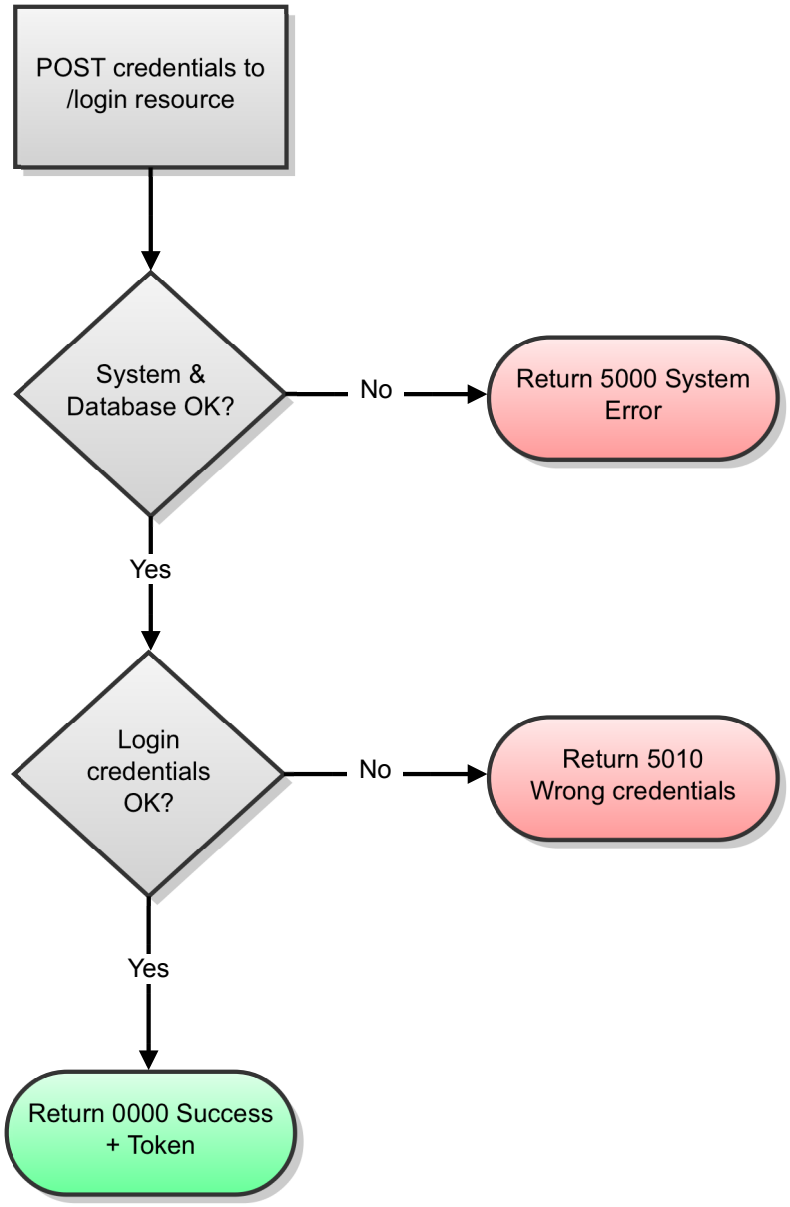
Field Name	Description	Example
status	Status code. 0000 is Successful login.	0000
message	Message related to the status code.	success
token	A limited-time security token of typically up to 90 characters. This token is passed with all other messages to authorize the instruction. It expires after 120 minutes.	RGn/ZsNK2Vos89BgIVsv9Ft-wzMZ8rZD4ELpXmWF1iw3Plw1-GKTf8R1O5y6VfYOgGh+wJRmuH-gb5Lz2VClKu1Q==

9.4. Note about the token format

The token is a Base64 version of a unique identifier that we create. Base64 strings can contain some characters (+ has commonly caused problems) that are mangled by some HTTP clients, since they are automatically URL-decoded. If you have spaces in your token, its because the client you use interpreted the plusses as URL-encoded spaces.

The token is case sensitive, and we don't actually return extraneous whitespace in our tags, the line-breaks in the examples are caused by the text wrapping.

9.5. Login Flowchart



9.6. Sample Login Responses

9.6.1. Login failure, incorrect credentials

Listing 1: Login failure, JSON

```
{"response":
  {"message":"Login Failure. Incorrect Username or Password."
  ,"status":5010}
}
```

Listing 2: Login failure, XML

```
<response>
  <status>5010</status>
  <message>Login Failure. Incorrect Username or Password.</message>
</response>
```

9.6.2. Login Success

Listing 3: Login success, JSON

```
{"response":
  {"message":"Login Successful."
  ,"token":"a26oenG5EuA9dsGINpY5RNIaiUNVjf/jY8bjubezHGvRmYKb+
     iVN6GlR8ikPgvVPF4yqh5H3gO+wuuC/bq8oKyoWqkFUZWMDAprjqONXj4w="
  ,"status":"0000"}
}
```

Listing 4: Login success, XML

```
<response>
<status>0000</status>
<message>Login Successful.</message>
<token>a26oenG5EuA9dsGINpY5RNIaiUNVjf/jY8bjubezHGvRmYKb+
   iVN6GlR8ikPgvVPF4yqh5H3gO+wuuC/bq8oKyoWqkFUZWMDAprjqONXj4w=</token>
</response>
```

10. Recharge

The /recharge resource is where you will submit recharges. HOTSOCKET then handles the PIN-less recharge to the mobile network operators.

10.1. Resource

Submit your ~~naughty pics~~ recharges here:
http://api.hotsocket.co.za:8080/recharge/
or
http://api.hotsocket.co.za:8080/test/recharge/ for testing.

10.2. POST Fields

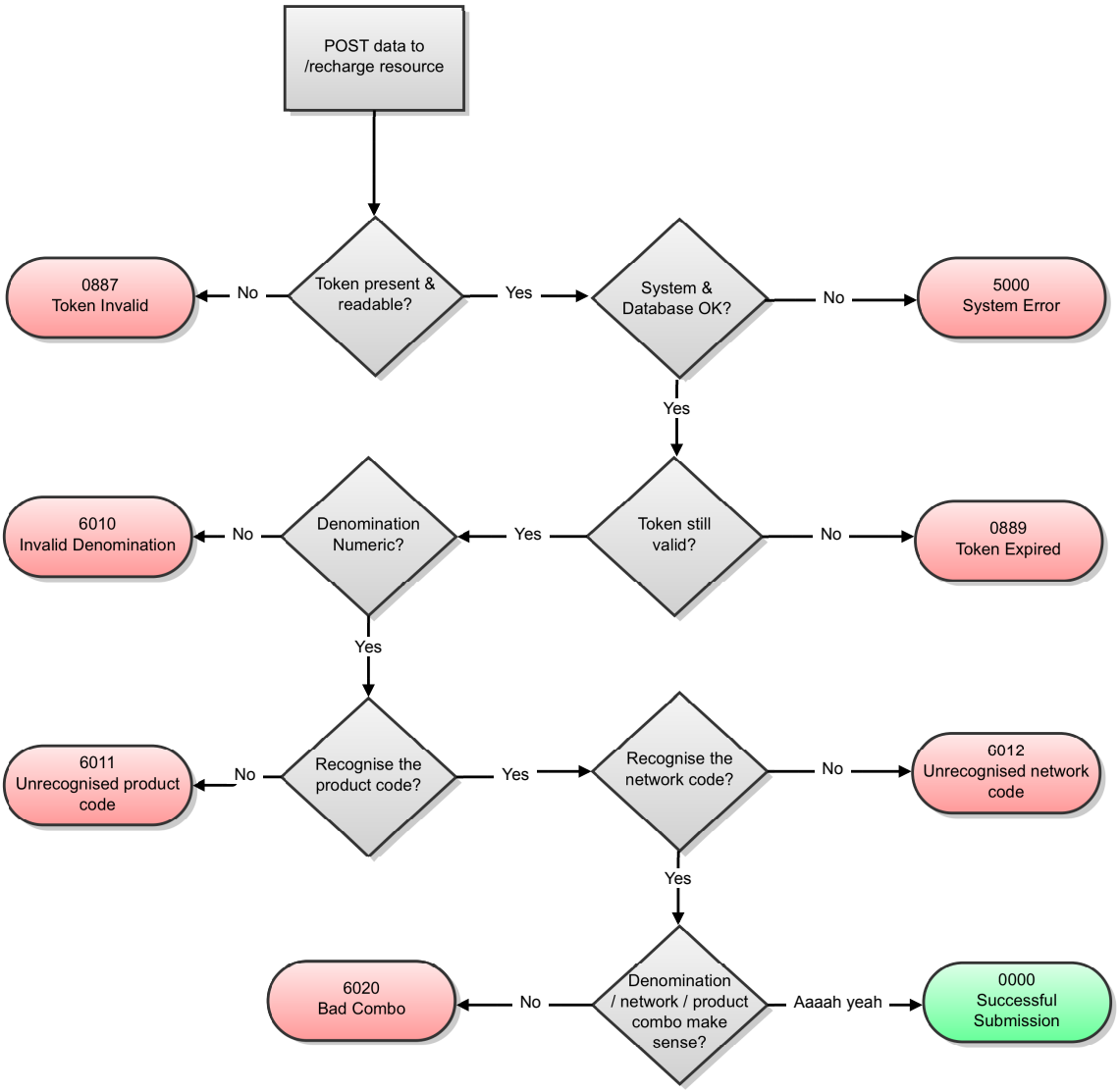
Field Name	Description	Example
token	Security token returned by the login.	RGn/ZsNK2Vos89BgIVsv9Ft-wzMZ8rZD4ELpXmWF1iw3Plw1-GKTf8R1O5y6VfYOgGh+wJRmuH-gb5Lz2VClKu1Q==
username	Your username. Max 25 chars, alphanumeric.	LeonTheProfessional
recipient_msisdn	The cell number of the SIM to receive the recharges. Format is E.164 but without the +.	27821231234
product_code	AIRTIME, DATA, or SMS.	DATA
denomination	Value (amount) of the product that should be loaded. Please refer to the valid denominations table to find valid combinations.	10
network_code	The network code. Network codes has a list.	VOD
reference	Your reference. This must be numeric. This must be unique. You will probably look the status up using this reference.	12345
notes	**Optional**. Further information to identify the transaction. 250 characters max.	Recharge for Bert
as_json	**Optional**. Send true if you want the results as JSON and not XML.	true

10.3. Response Fields

Field Name	Description	Example
status	Status code. 0000 means successfully submitted. - i.e. HOTSOCKET received the request and entered it correctly.	0000
message	Message related to the status code. Max 255 chars.	Success
error_id	Further error code if status is not 0000. Not always populated.	
error_message	Further error code if status is not 0000. Not always populated.	
serveport_ref	**Deprecated**. Our internal reference, rather just use the value you passed into reference when doing lookups. **Will be removed in future** - here for compatibility with Serveport.	112411
hotsocket_ref	The new name for serveport_ref, identical in value to it, but unlike serveport_ref, will be around in future.	112411

It is important to understand what 'successful submission' means - it only means your submission to HOTSOCKET was fine. HOTSOCKET will need to submit the recharge to the network, and you will still need to use /status to see if the actual recharge went through fine on the network side. 'Successful submission' is basically a Roger, Wilco from HOTSOCKET .

10.4. Recharge Flowchart



10.5. Example Responses

10.5.1. Successful submission

Listing 5: Successful recharge submission, XML

```
<response>
<status>0000</status>
<message>Successfully submitted recharge.</message>
<serveport_ref>1616</serveport_ref>
<hotsocket_ref>1616</hotsocket_ref>
</response>
```

Listing 6: Successful recharge submission, JSON

```
{"response":
  {"message":"Successfully submitted recharge."
  ,"serveport_ref":1617
  ,"hotsocket_ref":1617
  ,"status":"0000"}
}
```

10.5.2. Invalid Token

Listing 7: Invalid Token, XML

```
<response>
<status>0887</status>
<message>Token is invalid, please login again to obtain a new one.</
   message>
</response>
```

Listing 8: Invalid Token, JSON

```
{"response":
  {"message":"Token is invalid, please login again to obtain a new one.
     "
  ,"status":887}
}
```

10.5.3. Expired Token

Listing 9: Expired Token, XML

```
<response>
<status>0889</status>
<message>Token has timed out, please login again to obtain a new one.
</message>
</response>
```

Listing 10: Expired Token, JSON

```
{"response":
  {"message":"Token has timed out, please login again to obtain a new
     one."
  ,"status":889}
}
```

10.5.4. Malformed MSISDN

Listing 11: Malformed MSISDN, XML

```
<response>
<status>6014</status>
<message>Recipient MSISDN is malformed.</message>
</response>
```

10.5.5. Non-numeric characters in MSISDN

Listing 12: Non-numeric MSISDN, XML

```
<response>
<status>6013</status>
<message>Recipient MSISDN is not numeric.</message>
</response>
```

Listing 13: Non-numeric MSISDN, JSON

```
{"response":
  {"message":"Recipient MSISDN is malformed."
  ,"status":6014
  ,"error_id":"Recipient MSISDN format not recognised."}
}
```

10.5.6. Unknown Product

Listing 14: Unknown Product, JSON

```
{"response":
  {"message":"Unrecognized product code, valid codes are AIRTIME, DATA,
      and SMS."
  ,"status":6011}
}
```

10.5.7. Unknown Network

Listing 15: Unknown Network, JSON

```
{"response":
  {"message":"Unrecognized network code."
  ,"status":6012}
}
```

10.5.8. Network - Product - Denom combo not valid

Listing 16: Network, Product, Denom combo invalid, XML

```
<response>
<status>6020</status>
<message>Network code + Product Code + Denomination combination is
    invalid</message>
</response>
```

Listing 17: Network, Product, Denom combo invalid, JSON

```
{"response":
  {"message":"Network code + Product Code + Denomination combination is
      invalid"
  ,"status":6020}
}
```

10.5.9. Duplicate reference

Listing 18: Duplicate reference, XML

```
<response>
<status>6016</status>
<message>Reference must be unique.</message>
</response>
```

Listing 19: Duplicate reference, JSON

```
{"response":
  {"message":"Reference must be unique."
  ,"status":6016}
}
```

10.5.10. Non-numeric reference

Listing 20: Reference is not numeric, JSON

```
{"response":
  {"message":"Reference must be a numeric value."
  ,"status":6017}
}
```

10.6. Recipient MSISDN format

The MSISDN is the cell number that should receive this recharge. The format is E.164, with no non-numeric characters (that is, remove the leading plus too). 27821231234 is an example of the format.

Google open-sourced their libphonenumber library - it's great for working with phone numbers, and various versions in various languages exist (though your specific language might need some googling).

10.7. Error IDs

ID	Description
0	No Error.
1	Invalid recipient MSISDN. Use E.164 format.
2	Invalid Network. You sent an unrecognized network code.
3	Invalid (probably non-numeric) external reference number.
4	The note field contains invalid characters.
6	Invalid denomination. Refer to the valid denominations table.

11. Status Lookup

The /status resource returns the status of the recharge as returned by the mobile network operator. Status can handle your reference or the reference HOTSOCKET returned - just leave the other reference blank (it's XOR, so don't send both, you will get an error message).

11.1. Resource

http://api.hotsocket.co.za:8080/status/
or
http://api.hotsocket.co.za:8080/test/status/ for testing.

11.2. POST Fields

Field Name	Description	Example
token	Security token returned by the login.	RGn/ZsNK2Vos89BgIVsv9Ft-wzMZ8rZD4ELpXmWF1iw3Plw1-GKTf8R1O5y6VfYOgGh+wJRmuH-gb5Lz2VClKu1Q==
username	Your username. Max 25 chars, alphanumeric.	LeonTheProfessional
reference	The reference you passed to the recharge resource.	12345
hotsocket_ref	The reference we returned when you submitted a recharge. We recommend that you just always use your reference. **Serveport users** - code using serveport_ref will still work.	112411
as_json	**Optional**. Send true if you want the results as JSON and not XML.	true

11.3. Response Fields

Field Name	Description	Example
status	Status code. 0000 means successful status lookup.	0000
message	Message related to the status code. Max 255 chars.	Successful status lookup
recharge_status_cd	Status code of the recharge's status - see recharge status codes.	3
recharge_status	Extra text info related to the recharge_status_cd.	Successful
running_balance	Your purse balance at the time the recharge was submitted to HOTSOCKET .	100.50

11.4. Recharge Status Codes

Code	Description	Action
0	Submitted, not yet succesful.	Check again later, HOTSOCKET is waiting for network confirmation.
1	Pre-submission error. You will only see status 1 if have received an error when you tried to submit this to `/recharge`.	Fix problem, resubmit with new ID.
2	Failed.	See the list of messages for an explanation.
3	Success.	Have lunch, or similar.

Most recharges will take a few seconds to about a minute to become successful (i.e. sit around at status 0 and then go status 3). Recharges can infrequently be delayed on the network. Delays are officially indefinite, but typically resolve within three days in bad cases.

11.4.1. Messages returned

Code	Text
0	Recharge delayed.
0	Soon to be submitted
0	Submission error, will retry
0	Submitted to network, waiting for recharge response
1	Invalid Denomination
1	Invalid Product Code
1	MSISDN not understood
1	Recharge reference is not unique
1	Reference cannot be blank
1	Reference has to be a long numeric value
1	Unknown or invalid Network Code
1	You do not have sufficient funds for this transaction
2	A Serveport system error occured
2	MNO reports invalid MSISDN (not prepaid). You have not been billed for this.
2	MNO returned an unspecified error. You have not been billed for this.
2	SIM not registered for RICA. You have not been billed for this.
2	Submission error, retry count reached
3	Successful

Receiving invalid / not prepaid MSISDN means the mobile number operator did not recognise the MSISDN. Either the network was incorrect or (more common, if you get your MSISDNs from your clients), the client incorrectly assumed that prepaid bundles can be loaded on contract SIMs.

11.5. Example Responses

11.5.1. Recharge Successful

Listing 21: Status returning Successful Recharge, XML

```
<response>
<status>0000</status>
<message>Status lookup successful.</message>
<recharge_status_cd>3</recharge_status_cd>
<recharge_status>Successful</recharge_status>
<running_balance>0.0</running_balance>
</response>
```

Listing 22: Status returning Successful Recharge, JSON

```
{"response":
  {"message":"Status lookup successful."
  ,"recharge_status_cd":3
  ,"status":"0000"
  ,"running_balance":0
  ,"recharge_status":"Successful"}
}
```

12. Retrieving Lists of Recharges

The /statement resource returns a bunch of recharges that can be filtered by date, MSISDN, or product.

12.1. Resource

http://api.hotsocket.co.za:8080/statement/
or
http://api.hotsocket.co.za:8080/test/statement/ for testing.

12.2. POST Fields

Field Name	Description	Example
token	Security token returned by the login.	RGn/ZsNK2Vos89BgIVsv9Ft-wzMZ8rZD4ELpXmWF1iw3Plw1-GKTf8R1O5y6VfYOgGh+wJRmuH-gb5Lz2VClKu1Q==
username	Your username. Max 25 chars, alphanumeric.	LeonTheProfessional
start_date	Date from which you want a statement. YYYY-MM-DD format.	2015-01-01
end_date	End date for the statement. YYYY-MM-DD format.	2015-12-31
recipient_msisdn	**Optional**. Filter to only return statements for this recipient MSISDN.	27821231234
product_code	**Optional**. Optional filter to only return info associated with the given product code.	AIRTIME
as_json	**Optional**. Send true if you want the results as JSON and not XML.	true

12.3. Response Fields

Field Name	Description	Example
status	Status code. 0000 is successful statement lookup.	0000
message	Response message.	Successful statement lookup
line_item	Multiple line items, consisting of:	
request_date	Date that the request was sent through. YYYY-MM-DD hh:mm.	2015-01-21 17:14
recipient_msisdn	The cell number that received the recharge.	27791231234
external_reference	Reference supplied by you.	12345
note	Note, if supplied by you.	Ad hoc for Bert
status_date	Date the status was last updated for this recharge.	2015-03-11 12:34
status_desc	Description of the status.	Successful
denomination	Denomination.	1000
product_code	Product code.	AIRTIME
network_code	Network code.	TELKOM

12.4. Example Response

Listing 23: Statement, XML

```
<response>
<status>0000</status>
<message>Success.</message>
  <line_item>
    <request_date>2015-01-2217:19</request_date>
    <recipient_msisdn>+27791231234</recipient_msisdn>
    <external_reference>12345</external_reference>
    <note>Adhoc for Koos.</note>
    <status_date>2015-01-2214:30</status_date>
    <status_desc>Successful.</status_desc>
    <denomination>512</denomination>
    <product_code>DATA</product_code>
    <network_code>VOD</network_code>
  </line_item>

  <line_item>
    <request_date>2015-01-2217:25</request_date>
    <recipient_msisdn>+27820004321</recipient_msisdn>
    <external_reference>12346</external_reference>
    <note>Meh.</note>
    <status_date>2015-01-2217:26</status_date>
    <status_desc>Successful.</status_desc>
    <denomination>2900</denomination>
    <product_code>AIRTIME</product_code>
    <network_code>VOD</network_code>
  </line_item>
</response>
```

13. Balance Lookup

HOTSOCKET operates on a pre-funded model, so you can use the /balance resource to check how much funds you have left. You also get a running balance when submitting a recharge.

13.1. Resource

http://api.hotsocket.co.za:8080/balance/
or
http://api.hotsocket.co.za:8080/test/balance/ for testing.

13.2. POST Fields

Field Name	Description	Example
token	Security token returned by the login.	RGn/ZsNK2Vos89BgIVsv9Ft-wzMZ8rZD4ELpXmWF1iw3Plw1-GKTf8R1O5y6VfYOgGh+wJRmuH-gb5Lz2VClKu1Q==
username	Your username. Max 25 chars, alphanumeric.	LeonTheProfessional
as_json	**Optional**. Send true if you want the results as JSON and not XML.	true

13.3. Response Fields

Field Name	Description	Example
status	Status code. 0000 is successful balance lookup.	0000
message	Response message.	Balance lookup successful.
running_balance	Balance of your HOTSOCKET purse, in Rand.	42121.50

13.4. Example Responses

Listing 24: Balance, XML

```
<response>
  <status>0000</status>
  <message>Balance lookup successful.</message>
  <running_balance>483.0</running_balance>
</response>
```

Listing 25: Balance, JSON

```
{"response":
  {"message":"Balance lookup successful."
  ,"status":"0000"
  ,"running_balance":820}
}
```

14. Code Snippets

These are meant as very minimal starting points to connect to HOTSOCKET , and to be self-explanatory. We strongly suggest you use a built-in or third-party HTTP library for your language of choice. We have gone with the simple stuff in our examples.

14.1. PHP, basic login, print results (using cURL)

Listing 26: Basic Login, PHP

```
<?php
$ch = curl_init();
$params = 'username=your_username&password=your_password';

curl_setopt($ch, CURLOPT_HEADER, true);
curl_setopt($ch, CURLOPT_VERBOSE, true);
curl_setopt($ch, CURLOPT_URL, 'http://api.hotsocket.co.za:8080/login/')
    ;
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

//cheat to make cURL handle HTTPS:
//curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
//curl_setopt($ch, CURLOPT_SSL_VERIFYHOST,  2);

$result = curl_exec($ch);
curl_close($ch);
echo $result;
```

14.2. Python login, print results (httplib)

Listing 27: Basic Login, Python

```
import httplib, urllib

params = urllib.urlencode({'username':'myUsername', 'password':'
    myPassword'})
headers = {'Content-type':'application/x-www-form-urlencoded', 'Accept'
    :'text/plain'}

#Python isn't too strict about checking HTTPS stuff, so it's easy:
conn = httplib.HTTPSConnection('api.hotsocket.co.za:8080')
conn.request('POST', '/test/login', params, headers)
response = conn.getresponse()
print response.status, response.reason

data = response.read()
conn.close()
print data
```

14.3. Ruby login, print results

Listing 28: Basic Login, Ruby

```
require 'net/https'
require 'uri'

uri = URI.parse('https://api.hotsocket.co.za:8080')
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE
path = '/test/login'

# POST request -> logging in
data = 'username=myUsername&password=myPassword'
headers = {
  'Content-Type' => 'application/x-www-form-urlencoded'
}

resp, data = http.post(path, data, headers)

puts 'Code = ' + resp.code
puts 'Message = ' + resp.message
resp.each {|key, val| puts key + ' = ' + val}
puts data
```

14.4. Java, login, print results

Listing 29: Basic Login, Java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;

public class PostTest
{

    public static void main(String[] args)
    {
        try
        {
            String data = java.net.URLEncoder.encode("username", "ISO
               -8859-1") + "=" + URLEncoder.encode("myUsername", "ISO
               -8859-1");
            data += "&" + URLEncoder.encode("password", "ISO-8859-1") +
                 "=" + URLEncoder.encode("myPassword", "ISO-8859-1");
            // and so forth, with key value pairs
            // data += "&" + URLEncoder.encode("keyN", "ISO-8859-1") +
               "=" + URLEncoder.encode("valueN", "ISO-8859-1");
```

```
            java.net.URL url = new URL("http://api.hotsocket.co.za
                :8080/test/login");
            URLConnection conn = url.openConnection();
            conn.setDoOutput(true);
            OutputStreamWriter wr = new OutputStreamWriter(conn.
                getOutputStream());
            wr.write(data);
            wr.flush();

            // Get a response, hopefully
            BufferedReader rd = new BufferedReader(new
                InputStreamReader(conn.getInputStream()));
            String line;

            while ((line = rd.readLine()) != null)
            {
                //Process returned lines, uh, I mean, XML
                System.out.println(line);
            }

            wr.close();
            rd.close();
        }
        catch (UnsupportedEncodingException uee)
        {
            // URLEncoder.encode doesn't like you
            uee.printStackTrace();
        }
        catch (MalformedURLException mue)
        {
            //new URL couldn't be created
            mue.printStackTrace();
        }
        catch (IOException ioe)
        {
            ioe.printStackTrace();
        }
    }

}
```

15. Contact Us

15.1. Website

www.flickswitch.co.za

15.2. Email

hello@flickswitch.co.za

15.3. Johannesburg Address

Anchor Building, 1st Floor

The Media Mill

7 Quince Street, Milpark

2092

15.4. Cape Town Address

58 Wale Street

Cape Town

Western Cape, South Africa

8000

15.5. Telephone

+27 (0)87 943 7222

16. Appendix A - Networks and Products

16.1. Network codes

Code	Network	Available products
CELLC	Cell C	Airtime and data bundles.
MTN	MTN	Airtime and data bundles.
TELKOM	Telkom Mobile	Airtime and data bundles.
VOD	Vodacom	Airtime, data, and SMS bundles.

16.2. Product Code and Denomination combinations

16.2.1. Cell C

Product Code	Description	Denomination	Description	Price
DATA	DATA Bundle	25	Smartdata 25MB	R 6.00
DATA	DATA Bundle	50	Smartdata 50MB	R 10.00
DATA	DATA Bundle	100	Smartdata 100MB	R 19.00
DATA	DATA Bundle	300	Smartdata 300MB	R 55.00
DATA	DATA Bundle	500	Smartdata 500MB	R 85.00
DATA	DATA Bundle	1024	Smartdata 1GB	R 149.00
DATA	DATA Bundle	2048	Smartdata 2GB	R 245.00
DATA	DATA Bundle	3072	Smartdata 3GB	R 299.00
DATA	DATA Bundle	5120	Smartdata 5GB	R 399.00
DATA	DATA Bundle	10240	Smartdata 10GB	R 549.00
DATA	DATA Bundle	20480	Smartdata 20GB	R 1099.00

16.2.2. MTN

Product Code	Description	Denomination	Description	Price
DATA	DATA Bundle	5	MTN 5MB	R 4.00
DATA	DATA Bundle	20	MTN 20MB	R 12.00
DATA	DATA Bundle	50	MTN 50MB	R 25.00
DATA	DATA Bundle	100	MTN 100MB	R 35.00
DATA	DATA Bundle	300	MTN 300MB	R 85.00
DATA	DATA Bundle	500	MTN 500MB	R 105.00
DATA	DATA Bundle	1024	MTN 1GB	R 160.00
DATA	DATA Bundle	2048	MTN 2GB	R 260.00
SMS	SMS Bundle	50	50 SMS	R 17.00
SMS	SMS Bundle	100	100 SMS	R 30.00
SMS	SMS Bundle	200	200 SMS	R 50.00
SMS	SMS Bundle	500	500 SMS	R 114.00
SMS	SMS Bundle	2000	2000 SMS	R 420.00

16.2.3. Telkom Mobile

Product Code	Description	Denomination	Description	Price
DATA	DATA Bundle	25	25MB All Networks	R 7.25
DATA	DATA Bundle	50	50MB All Networks	R 14.50
DATA	DATA Bundle	100	100MB All Networks	R 29.00
DATA	DATA Bundle	250	250MB All Networks	R 39.00
DATA	DATA Bundle	500	500MB All Networks	R 69.00
DATA	DATA Bundle	1024	1GB All Networks	R 99.00
DATA	DATA Bundle	2048	2GB All Networks	R 139.00
DATA	DATA Bundle	3072	3GB All Networks	R 199.00
DATA	DATA Bundle	5120	5GB All Networks	R 299.00
DATA	DATA Bundle	10240	10GB All Networks	R 499.00
DATA	DATA Bundle	20480	20GB All Networks	R 899.00
DATA	DATA Bundle	51200	50GB All Networks	R 1799.00

16.2.4. Vodacom

Product Code	Description	Denomination	Description	Price
DATA	DATA Bundle	15	MyMeg 15	R 9.00
DATA	DATA Bundle	30	MyMeg 30	R 12.00
DATA	DATA Bundle	100	MyMeg 100	R 29.00
DATA	DATA Bundle	250	MyMeg 250	R 59.00
DATA	DATA Bundle	500	MyMeg 500	R 99.00
DATA	DATA Bundle	1024	MyGig 1	R 149.00
DATA	DATA Bundle	2048	MyGig 2	R 249.00
DATA	DATA Bundle	3072	MyGig 3	R 299.00
DATA	DATA Bundle	5120	MyGig 5	R 399.00
DATA	DATA Bundle	10240	MyGig 10	R 599.00
DATA	DATA Bundle	20480	MyGig 20	R 999.00
SMS	SMS Bundle	20	20 SMS Bundle	R 10.00
SMS	SMS Bundle	50	50 SMS Bundle	R 25.00
SMS	SMS Bundle	100	100 SMS Bundle	R 33.00
SMS	SMS Bundle	150	150 SMS Bundle	R 49.00
SMS	SMS Bundle	200	200 SMS Bundle	R 45.00
SMS	SMS Bundle	300	300 SMS Bundle	R 67.50
SMS	SMS Bundle	500	500 SMS Bundle	R 112.50
SMS	SMS Bundle	1000	1000 SMS Bundle	R 225.00
SMS	SMS Bundle	1500	1500 SMS Bundle	R 337.50
SMS	SMS Bundle	2000	2000 SMS Bundle	R 450.00