RouteLink®

SPECIFICATIONS

RouteLink API 2.0

October 02, 2018

Version: 0.1

somos.com
o 844.HEY.SOMOS
f 732.514.6723
P.O. Box 8122
Bridgewater, NJ 08807-8122

RouteLink API

- For general information about this document, please call or text the Help Desk at 844.HEY.SOMOS (844.439.7666), Option 1.

TRADEMARK ACKNOWLEDGEMENTS

SECURITY CLASSIFICATION – SOMOS EXTERNAL

Property of Somos, Inc.

Revision History

Revision History		
Date	***Version***	***Description***
October 02, 2018	0.1	Initial version of API 2.0

Table of Contents

1. Audience

The audience for this document is the development team using the RouteLink API 2.0 to connect to the RouteLink service. This document assumes the reader is already familiar with the SMS/800, Toll Free Numbers, REST, JSON, and message digests.

2. Introduction

This document describes the API to download Call Routing Number (CRN) routing information from the RouteLink Server. CRNs are frequently referred to as Toll Free Numbers (TFNs) and are often used interchangeably in this document. In general, RouteLink contains an Event Queue with First-In-First-Out events such as "add TFN" and "delete TFN" to be downloaded to the RouteLink Customer (the RouteLink client), in order.

2.1 Event Queue Order

The RouteLink Client requests events to be downloaded. The events MUST be downloaded and processed in FIFO order to guarantee the accuracy of the client's database.

3. API

The interface between the RouteLink Server and the RouteLink Client is REST/JSON. The client sends HTTPS GET and POST requests to the server. The server responds to the GET or POST with a JSON encoded response.

The following table enumerates the possible URLs. Each URL is preceded by "/routelink/v2".

API URLs	
Url	***Comments***
/download	Download events from the Event Queue.
/initialize	Set the event queue back to an initial download.
/audit	Reply to an audit request from the RouteLink application
/refreshToken	Request a new token with a new expiration date.
/cpr?sha1=	Request JSON cpr by passing sha1 value which is required

IMPORTANT: Due to the large amount of data exchanged between RouteLink® and the client application, enabling gzip compression in the transport layer is strongly recommended. Compression is enabled by the client on a per-transaction basis. Each request sent by the client must indicate support for gzip compression by adding the appropriate information to the header. Specifically, the request header must state the client can accept gzip encoding via the header parameter: "Accept-Encoding: gzip" as reflected in the wget examples below.

3.1 Download Request

The following is an example REST/JSON download request:

```
GET /routelink/v2/download HTTPS/1.1
Authorization: Bearer c0d97b52-35d9-32c2-a37d-6126a186a844
```

There are several tools that can be used to aid in the initial development of the download. Below is an example that can be used in a UNIX environment.

The wget tool can issue a download like this example:

```
wget --header="Accept-Encoding: gzip" --header="Authorization: Bearer
c0d97b52-35d9-32c2-a37d-6126a186a844" https://api-
routelink.somos.com/routelink/v2/download
```

More generally the wget command is using the access token as follows:

```
wget --header="Accept-Encoding: gzip" --header="Authorization: Bearer <insert
your access token here>" https://api-
routelink.somos.com/routelink/v2/download
```

The download request can download several messages at once. The maximum number of messages is 5000 at one time. To download a specific number of messages, include the number of messages at the end or the URL as follows:

```
wget --header="Accept-Encoding: gzip" --header="Authorization: Bearer <insert
your access token here>" https://api-
routelink.somos.com/routelink/v2/download/1000
```

In the above example, the client is requesting 1000 messages. If the /1000 portion of the URL is not included, RouteLink will return a default number of messages to the client. This optional message count is a maximum number of messages the client is requesting so RouteLink will return up to that number of updates, depending on the number of available updates for the client.

The client SHOULD also send to RouteLink the last message index that it received (see message response below). The following example demonstrates the correct syntax:

```
wget --header="Accept-Encoding: gzip" --header="Authorization: Bearer <insert
your access token here>" https://api-
routelink.somos.com/routelink/v2/download/1000?lastIndex=78233
```

In the above case, RouteLink will return 1000 messages to the client where the id is greater than 78233. The id value ranges from 0 to 2^63 -1 (9,223,372,036,854,775,807). Although the lastIndex parameter is not required, it allows RouteLink to verify that last set of messages received by the client.

3.2 Download Response

The following is an example of a JSON download response from the RouteLink that includes 3 events in a JSON array. The first and last are *add* events where one includes the optional CPR.

```
{
    "events":[
```

```
            {
               "action":"add",
               "crn":"8005001212",
               "ror":"ROR01",
               "sha1":"<40 chars>",
               "cpr":null,
               "id":1000
            },
            {
               "action":"delete",
               "crn":"8006001212",
               "id":1001
            },
            {
               "action":"add",
               "crn":"8007001212",
               "ror":"AR26",
               "sha1":"<40 chars>",
               "cpr":"<variable length JSON CPR>"
               "id":1002
            }
         ]
      }
```

Field Descriptions		
Field	***Range***	***Comments***
events	an ordered array	A list that MUST be processed in order to maintain the accuracy of the client's local DB.
action	“add”, “delete”, "cpr" or "audit_request"	Add or delete a CRN from the client DB. A redundant add (for an already active CRN) must be treated as a replace of the CRN’s data, such as replacing the ROR (and the CPR, if present).
crn	exactly 10 printable digits	Usually a 10 char CRN (e.g. “8005551212”). But for template CPRs, this value can start with the number "0".
ror	1 to 5 printable chars	The RespOrg ID associated with the CRN. This is *always* present for an "add" and *never* present for a "delete".
sha1	exactly 40 printable hex digits	The SHA-1 hash of the associated JSON CPR. A SHA-1 hash is *exactly* 40 characters long, as printable hex. For example: "E076B32E287452057362532D38E239F9462D3AF4". The field is *always* present for an "add" and *never* present for a "delete".
cpr	characters	The Call Processing Record presented as a JSON construct.

		This field is only present for an "add" and even then it is optional and only sent the first time the CPR SHA-1 hash is encountered. The client is required to save this value. The next time the same SHA-1 hash occurs, null is sent, and the client should refer to the earlier CPR value. See Section "8.1 JSON Call Processing Record Structure" for a detailed description of the CPR layout in its JSON format.
id	number It may also be set to null (see description)	This value represents an index used by RouteLink to track various data types. It is provided to an API client to allow it to request specific messages that may have been missed. An API client should provide the last (highest) id value it received from RouteLink as the lastIndex field on the next request to RouteLink. This value may be null when RouteLink is performing an audit (initial audit event will have id which should be sent in subsequent download request after successful audit) with an API client or while requesting a CPR.

3.2.1 Download Responses with No Events

If the RouteLink Server has no events to send in the response, or if the RouteLink Server is currently unable to reply with events for any reason, then an empty event list is sent. The client must assume that there is no more data to currently download, and try again later.

The JSON for an empty event array is:

```
{"events" : [] }    // the empty event array
```

3.2.2 CRN "add", SHA-1, and the Optional CPR

When an *add* is encountered, the entire JSON CPR is hashed using the SHA-1 algorithm. The algorithm produces a 20-byte hexadecimal value that is converted into 40 printable bytes and sent in the "sha1" parameter of the "add" event.

When a unique SHA-1 hash is encountered for the first time by the RouteLink Server, it sends the full JSON CPR as well as its SHA-1 hash to the RouteLink Client.

The client MUST keep track of both the SHA-1 hash and the JSON CPR in its local DB for later audits. Also, when the RouteLink Server encounters the same SHA-1 hash in future "add" events, the RouteLink Server only sends the SHA-1 hash given the likelihood the client already has the full CPR (i.e. sending the CPR would be redundant). The advantage of this approach is that bandwidth for both the client and the server is significantly reduced since a large percentage of CPRs are re-used across CRNs. If, for any reason, the client does not have a matching CPR for the SHA-1 it can be requested via the '/cpr?sha1=' API.

3.3 Refresh Token Request

Each client token has an associated expiration date. Once the token expires, the token will be rejected by RouteLink. The following is an example REST/JSON refresh token request:

```
GET /routelink/v2/refreshToken HTTPS/1.1
Authorization: Bearer c0d97b52-35d9-32c2-a37d-6126a186a844
```

Note that the token inside the request is the "old" token about to expire (the one you are asking to refresh). This command fails if your token has already expired. If that happens, you must manually use the RouteLink web site to acquire a new token.

3.4 Refresh Token Response

The response to a refresh token request contains the new token. Upon receiving the response, the old token is invalid and only the new token can be used for any future API requests.

The following is an example JSON token refresh response from RouteLink.

```
{
    "token":
     "c0d97b52-35d9-32c2-a37d-6126a186a844",
    "expiration":"01/09/2017 15:56:21"      // always in GMT
}
```

Field Descriptions		
Field	***Range***	***Comments***
token	Printable String	The new token. The former token is no longer valid.
expiration	MM/DD/YYYY HH:mm:ss	The expiration timestamp of the new token. This value is *always* in GMT. The HH:mm:ss portion is in 24-hour format.

4. Initialization

When a RouteLink Client registers with the RouteLink service, the RouteLink Server resets the client's status to download from beginning. Note that the CPR data will be sent first as a "cpr" event followed by the CRN data as "add" events. The CPR records contain a hash value and any

CRN that uses that CPR will also contain the same hash value in its "add" event record. Once the initial "add" events are downloaded, all subsequent downloads are changes in CRN state ("add" and "delete" events). The initial load is composed of millions of events, but this initialization only has to occur once for the new client.

In the case where the client has an unrecoverable failure, it may need to reset all of its data and "start over". It is left to the RouteLink Server discretion to refuse an unexpected initialization request (for example the RouteLink Server could limit initializations to being accessible only after manual intervention indicates it is allowed for a client).

The client communicates an initialize request to the server using the following URL:

/routelink/v2/initialize

The RouteLink Server can issue a success or failure response. The failure response follows the same rules as presented in the Error Detection section of this document. The success JSON response is as follows:

```
{ "result": "success" }    // See Error Detection section for errors
```

Failure responses are left to the RouteLink Server's discretion but could include a rejection due to too many initializations being requested. The error message format is described in ***Section 6 – Error Detection***. During the "official" certification process, a full download of all data must be performed and requests for a single NPA will result in an error response.
A success reply will initiate logic at the RouteLink server to prepare the download data for the client. The following is an example steps that make up the initialize process.

1. The client clears its DB of all CRNs.
2. The client sends the initialize request.
3. The server resets the client's status to begin the download.
4. The client should periodically call the /routelink/v2/download URL and data will be sent once it is available.
    - If the client sends the lastIndex parameter as part of the download command, use a value of zero following the call to initialize.
5. The client, after downloading all the "cpr" and "add" events from its event queue, continues to receive new download events from the ongoing events that occurred during initialize process.
    - When the client consumes the last record from the initialization and starts receiving real-time updates, a large jump in the lastIndex value is expected (i.e. greater than 1 million)

Note: we are not taking the snapshot of DB for initializing the customer. Real time updates happen on RL during client's initialize process. We suggest client to have the referential integrity between crn's sha and cpr's SHA-1 so you will know if there is a new SHA-1 value during the initialize process.

If you find SHA-1, but there is no CPR in your db, you can either reject that message or use /cpr api to get the latest CPR for that sha1 value. Refer to sec 6 for /cpr details
If you reject the message, make sure you are consuming it through live updates. While

consuming live updates, if you see CPR value in your response, that indicates a new CPR. If CPR is null, that shows it's already existing.

If the RouteLink Server fails to initialize for any reason, it returns a failure response. The client is free to retry later if the failure is temporary. Please refer to the Error Detection section of this document to determine if an error is temporary or permanent.

4.1 Initialization Message Details

The following section describes the JSON format of the "cpr" message. Once all of the CPR messages are downloaded by the API client, "add" events for each call routing number will follow. The client should expect to receive more than 250,000 "cpr" messages and over 40 million "add" messages. The client may download up to 1000 CPR messages in a single transaction.

```
{
   "events":[
   {
      "action":"cpr",
      "sha1":"<40 chars>",
      "cpr":"<variable length JSON CPR>",
      "id": 1000
   },
   {
      "action":"cpr",
      "sha1":"<40 chars>",
      "cpr":"<variable length JSON CPR>"
      "id": 1001

    } ] }
```

5. Audit

The RouteLink Server maintains the master CRN/CPR database. Each downstream RouteLink Client therefore has an effective copy of the master database by downloading the add and delete events as they are made available by the server. As discrepancies are found events are placed on the queue and the RouteLink Client downloads the corrective event.
List of discrepancies:

- CRN missing in RouteLink client
- CRN present in RouteLink client but not in RouteLink server
- Mismatched ROR
- Mismatched CPR

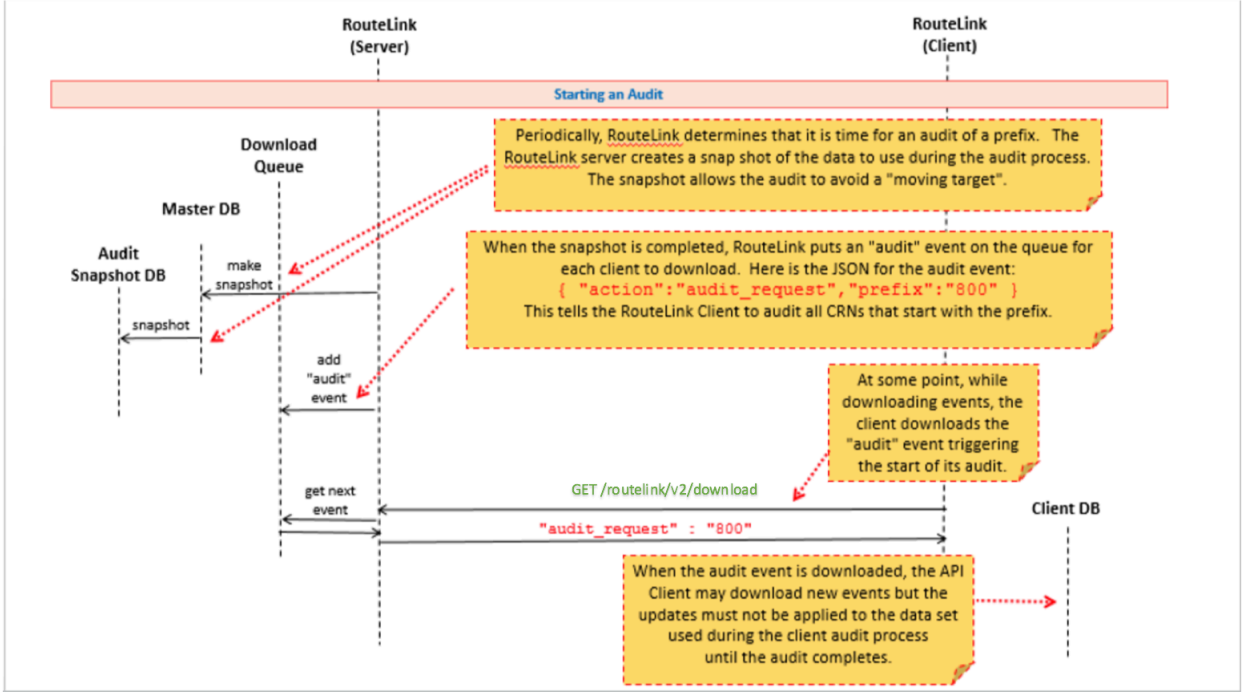
5.1 Audit Message Flows

The sections that follow describe the message flow for the audit for success and failure conditions. Lastly, this chapter enumerates the specific request and response messages used by the audit.

5.1.1 Starting an Audit

The RouteLink Server decides when an audit is necessary by placing an audit event for a specific prefix on the client's event queue. The following diagram describes the events and flow of messages that occur when a prefix audit for 800 is started. It's important to understand that there

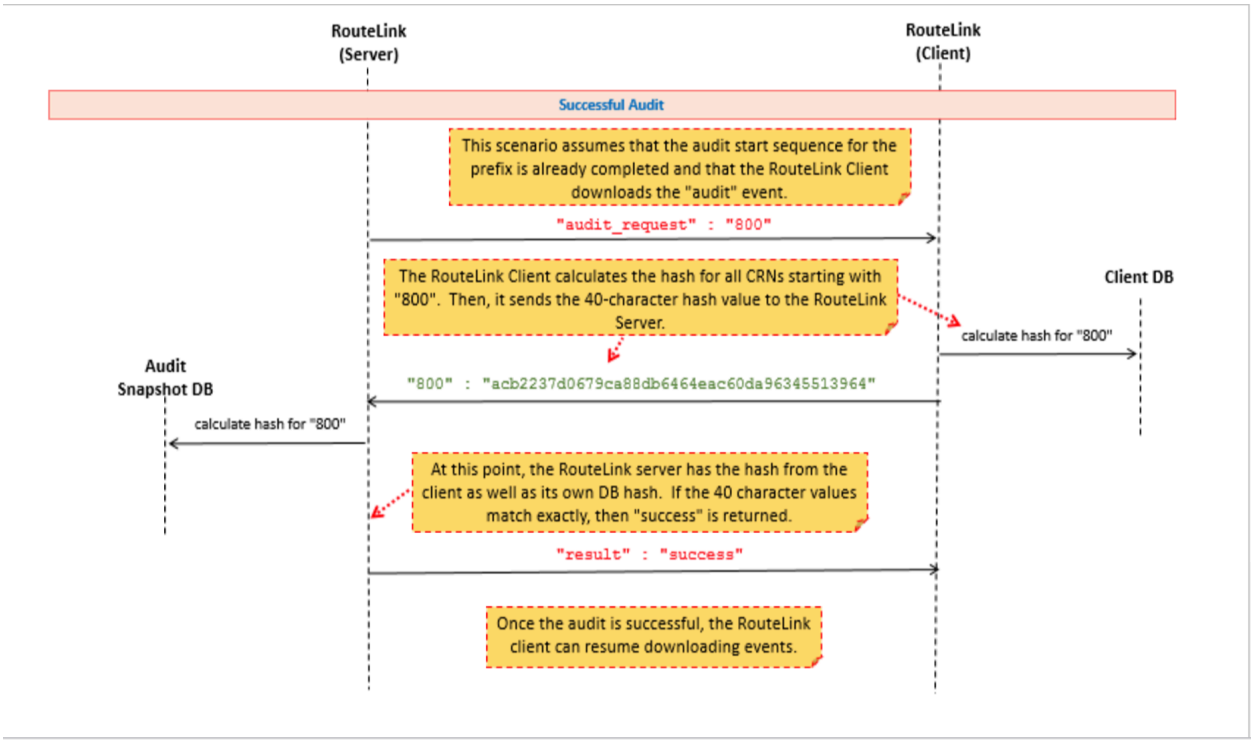
will be an audit for each 8xx prefix that is valid at the time of the audit. Currently these prefixes include 800, 833, 844, 855, 866, 877, 888, and 0. Numbers starting with a 0 denote template records. Please note this list will be expanded as new toll-free prefixes are added such as 822.



5.1.2 Description of a Successful Audit

The following message flow diagram shows the messages for the case when the audit of a prefix is successful. Only 3 messages are required in this best case.

The term, calculate hash, is used to indicate that a SHA-1 hash (40 printable hex digits) is generated from the data stored locally. The specific method for creating the hash is detailed in a later section.



The diagram points out that both the sever and the client must use databases that are not changing during the audit (to avoid auditing a moving target). When the audit event arrives at the client through an event download, the two databases should, theoretically, be identical. The databases are compared by having both sides calculate the hash for the given prefix.

The “prefix” in the audit indicates to the algorithm to process all CRNs that start with that prefix. In SQL terms, the SELECT can be thought of as using a "where" clause like (in this example the prefix was 800):

```
“WHERE (crn LIKE '800%')”.
```

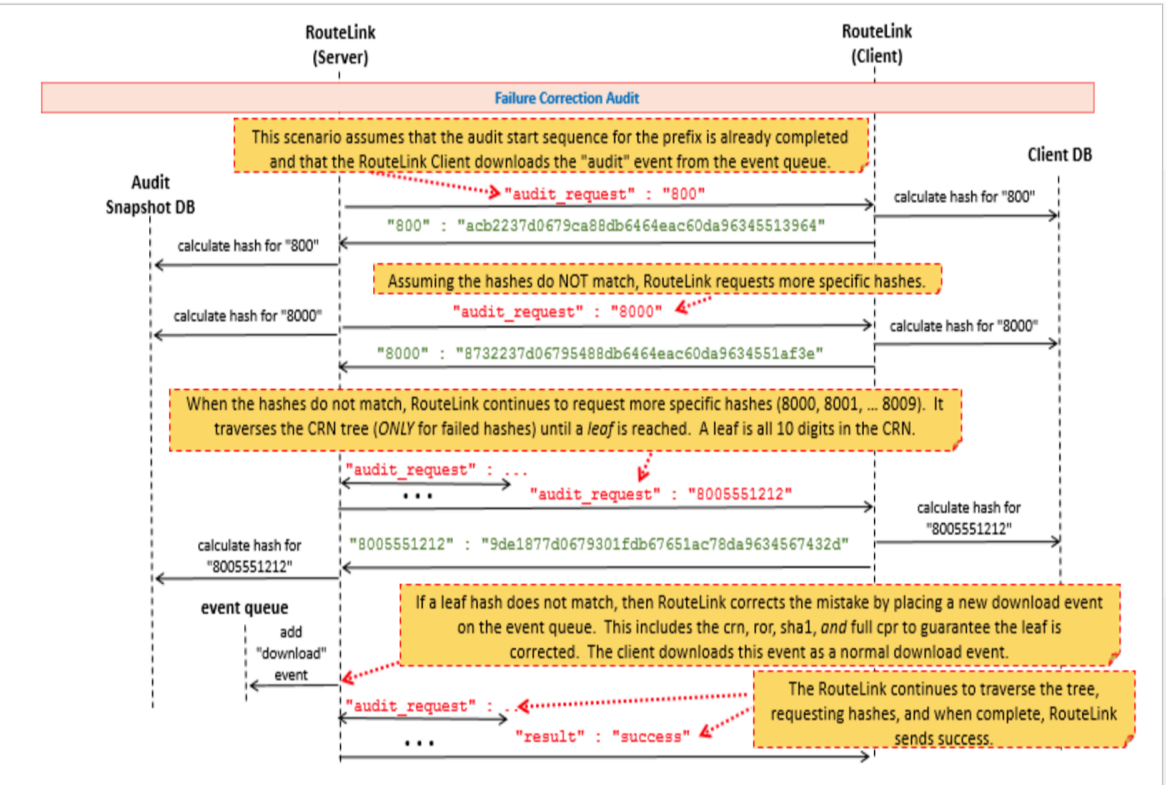
The server will request a hash value over a given prefix using the audit request message and the client must calculate the hash and return the result to the server using an audit reply message. The audit reply message is sent to the server using an HTTPS POST. An example of sending an audit reply using wget is as follows:

```
wget --header="Content-Type: application/json" --header="Authorization: Bearer
<access token>" --post-data='{"action":"audit_reply","prefix":"800","sha1":
"<40 character hash value>"}' https://api-
routelink.somos.com/routelink/v2/audit
```

The server compares the client's hash to its own hash to see if the two match. Of course, the calculation of the hash must be done using data fed into the algorithm in the exact same order by both sides. There is no requirement that the two sides use the same DB schema – but, the data must be fed into the SHA-1 algorithm in the same order. The data includes the CRNs, RORs, and CPR hashes (not the CPRs themselves). The hash calculation and the order of the data is described in more detail later.

5.1.3 Description of an Audit Correction

A failed audit, in this context, means that the initial hashes did not match. This will trigger further messages to find and correct the mistakes. Once all mistakes are corrected, the final message indicates success for that prefix. The diagram below shows how the client's DB is corrected when a failure is detected.



The following steps help explain the above diagram:

1. The audit event that is downloaded to start the prefix audit, is JSON:
   `{ "action":"audit_request", "prefix":"800", "id": 10000 }`
2. The client sends the hash for all CRNs starting with 800 in JSON similar to:
   `{ "action":"audit_reply", "prefix":"800", "sha1": "acb2237d0679ca88db6464eac60da96345513964" }`
   The 40 character SHA1 hash above is an example and its algorithm is described later.
3. The server performs the same hash algorithm for the 800 prefix on its Audit Snapshot DB.
4. The server compares the 2 hashes. If they don't match, then the server begins to traverse the CRN "tree". Each successive digit of a 10-digit CRN is a node in the tree with 10 branches. The server asks the RouteLink Client for hashes like "8000", "8001", … "8009" using more "audit" messages containing more digits. When the server receives each response from the client, it compares the hash to its own hash for that node. If the DBs are nearly the same, but not exact, almost all hashes will match. The server does not pursue any branches that match, so elimination of large portions of the tree happens quickly. The server continues to audit only the failed hashes. Each time the server requests an audit, it provides the starting CRN value to hash. A deeper example of a starting value would be "80055", thus telling the client to

calculate the hash using an SQL statement "where" clause similar to `"WHERE (crn LIKE '80055%')"`.

5. If the hash for a fully qualified, 10-digit CRN (a "leaf") fails to match, the server adds a new download event to the event queue for that CRN. The event *always* contains all the information (crn, ror, sha1, and full cpr). The client downloads the event, as a normal part of its download of CRN data.
6. When the server traverses all failed hash branches for the prefix, a success message will be returned:
   ```
   { "result" : "success" }
   ```

The server is solely responsible for directing the audit and traversing the CRN tree. The client simply responds to the audit requests with the hash of the requested CRN, which could include any number of digits up to a full 10-digit number. The client need not be aware of the tree traversal process, as it could change.

5.1.4 RouteLink Server Actions on Hash Failure

When the server encounters a failed hash, it begins a traversal of the database to determine the failed TFNs. Rather than traversing tens of millions of TFNs linearly, a more efficient algorithm is employed, where the server requests hashes of more specific, partial, CRNs. For example, the server requests the hash of a partial CRN of "8000", "8001", ... "8009" to narrow down its search. The client does not have to determine where the failure occurs, that is the responsibility of the server. The client only needs to be prepared to respond with any partial CRN hashes requested by the server. The partial CRN could have any length from 1 to 10 digits.

5.1.5 Input to Hash Algorithm

To obtain a hash of a partial CRN set of data, the input stream to the hash algorithm can be thought of as a file. *A file is not at all required*, but it is useful for descriptive purposes below. The input MUST match the following format, byte for byte, or the hashes calculated by the client and server will not match.

The format of the file (or an input stream) is as follows:

```
<CRN>,<ROR>,<SHA1>\n
<CRN>,<ROR>,<SHA1>\n
...and so on, one line per active CRN...
<CRN>,<ROR>,<SHA1>\n
```

The following points are required by the hash calculation:

1. The number of lines depends on the number of CRNs in the query. A partial CRN of "800" results in one line for every CRN with a prefix of 800.
2. The input MUST be ordered by the CRN, ascending.
3. The CRN, ROR, and SHA1 are printable strings like `"8005551212"`, `"AM123"`, and `"acb2237d0679ca88db6464eac60da9634d51396a"`.
4. There is a comma (no spaces) between the fields on each line.
5. Each `[CRN,ROR,SHA1]` combination ends with a single carriage return character, `\n`. (Not `\r\n`).

6. Like all other lines, the last line has a CRN, ROR, and SHA1, followed by a single terminating \n
   character.
7. The SHA-1 hash is sent in lower case hexadecimal.
8. If a request for a partial CRN results in no data (no matching CRNs), then the file (or input stream) is empty. The hash of an empty stream is:

   ```
   da39a3ee5e6b4b0d3255bfef95601890afd80709
   ```

   This can be demonstrated on a linux host as follows:

   ```
   $ echo -n "" | sha1sum
   da39a3ee5e6b4b0d3255bfef95601890afd80709      -
   ```

The following statement is an example MySQL query run on a linux host. This query can be used as a guide for creating a file formatted with the above rules. The query makes assumptions about the DB schema, so modify it as needed.

```
$ echo "SELECT CONCAT_WS(',',    crn, ror, sha1)     \
      FROM               myCrnTable    \
      WHERE (            crn LIKE '800%')    \
      ORDER BY                         crn"   \
      |      mysql MyDB -u MyUser -p MyPwd    \
      --disable-column-names                  \
      > myfile.txt
```

The following file is an example (SampleAuditFile1.txt) for the partial CRN “866449874”. Since the partial CRN has only 9 digits the hash is for the range “8664498740” .. ‘8664498749”. For this example, it is assumed that all CRNs have the same ROR and CPR SHA1. It is also assumed that there are only 4 active CRNs in this range: the CRNs ending in 1, 3, 7, and 9:

```
8664498741,AM467,acb2237d0679ca88db6464eac60da96345513964
8664498743,AM467,acb2237d0679ca88db6464eac60da96345513964
8664498747,AM467,acb2237d0679ca88db6464eac60da96345513964
8664498749,AM467,acb2237d0679ca88db6464eac60da96345513964
```

This file becomes input to the SHA-1 hash algorithm. The reader is encouraged to create the above file and apply a SHA-1 hash generator to it to confirm the hash value below. There are several online hash generators available on the web (google “sha-1 generator”). The correct hash for the above file is:

```
074d2a57e223dcf033cd44d14242036912c3ea8a
```

If you are working in a Linux environment, the following command may be available:

```
$ sha1sum SampleAuditFile1.txt
074d2a57e223dcf033cd44d14242036912c3ea8a      SampleAuditFile1.txt
```

5.2 Audit Message Details

The following sections describe the JSON details of each message discussed above.

5.2.1 Audit Request from RouteLink Server

The server is the only initiator of audit requests. The server initiates an audit for a prefix by loading an audit event onto the client's event queue. The client downloads this event and as a result must start the audit. Only the first audit request will contain an id value.
The following is an example event that starts the audit:

```
{ "action" : "audit_request", "prefix" : "800", "id" : <index> }
"800" indicates one of 11 prefixes that can be audited
```

"id" from above request should be sent in subsequent download request after audit is done.

The server also provides other partial CRNs for the audit, besides "800". The JSON for those messages is the same as the above, where "800" becomes more specific, partial, CRNs as the server traverses the CRN tree if it deems it necessary.
For example:

```
{ "action" : "audit_request", "prefix" : "866", "id" : <value> }
      // "WHERE (crn LIKE '866%')"
{ "action" : "audit_request", "prefix" : "8660", "id" : null }
      // "WHERE (crn LIKE '8660%')"
{ "action" : "audit_request", "prefix" : "8661", "id" : null }
      // "WHERE (crn LIKE '8661%')"
```

5.2.2 Audit Responses from RouteLink Client

The client is the only sender of audit reply. It issues a response containing the hash of the partial CRN indicated in the audit request. The API Client sends this message to RouteLink URL endpoint /routelink/v2/audit using an HTTPS POST.
The JSON of an example response to an audit of "800" is like:

```
{ "action":"audit_reply",
  "prefix":"800",
  "sha1": "acb2237d0679ca88db6464eac60da96345513964"
}
```

All CRNs for prefix 800

The JSON of an example response to an audit of "8662561" (thereby processing 8662561000..8662561999) is like:

```
{ "action":"audit_reply",
  "prefix":"8662561",
  "sha1": "3de1247d07398078db6487230a0da9634583096c"
}
```

5.2.3 Audit Completion

The RouteLink Server is the only sender of the success message. The server indicates that it is done with the audit for a prefix with this event:

```
{ "result" : "success" }
```

When the client receives this downloaded message, it can assume any corrections were made during the audit by placing new download events on its event queue.

5.3 Hash Mismatch Troubleshooting

There are scenarios where file contents may appear to be identical but the hash calculation is resulting in mismatched values.
In the example below there are non-printable characters present that can't be seen with a file diff.

```
$ diff SampleAuditFileRouteLink.txt SampleAuditFileCustomer.txt
2c2
< 8005551213,QR760,bdc1126e1780e076cde352f37e0fb48823605a78
---
> 8005551213,QR760,bdc1126e1780e076cde352f37e0fb48823605a78
```

A tool to generate a hex dump of the database file can be used to determine non-printable characters. Comparing the hex dump files between the RouteLink Client and RouteLink Server can highlight the issue.
Contact Somos support for a RouteLink Server database prefix file.
For example, the Linux utility od was used on the above files. The bytes in red are the hexadecimal values of each byte in the file. The difference between the two files is highlighted to show there is an extra space at the end of the second line in the customer output file.

```
$ diff  RLClient-HEX.txt RLServer-HEX.txt
8c8
 < 0000160   61   37   38   0a
               a    7    8   \n

 ---
 > 0000160   61   37   38   20   0a
               a    7    8        \n
```

6. CPR API

6.1 /cpr api request

/cpr is an optional api.
The following is an example REST/JSON CPR request:

```
GET /routelink/v2/cpr?sha1=93201524444RSRRKR1 HTTPS/1.1
Authorization: Bearer c0d97b52-35d9-32c2-a37d-6126a186a844
```

The wget tool can issue a cpr request like this example:

```
wget --header="Accept-Encoding: gzip" --header="Authorization: Bearer
c0d97b52-35d9-32c2-a37d-6126a186a844" https://api-
routelink.somos.com/routelink/v2/cpr?sha1=93201524444RSRRKR1
```

sha1 is required field.

6.2 /cpr api response

The response to a cpr request contains the JSON cpr and sha.

The following is an example JSON cpr response from RouteLink.


```
{
     "id":null,
     "sha1":"81a642c51a313314f828a0a8944e3c20be914f66",
     "cpr":"{\"node\":{\"type\":\"npa\",\"branches\": [{\"values\": [\"204\",\"226\",\
"236\",\"249\",\"250\",\"289\",\"306\",\"343\",\"365\",\"367\",\"403\",\"416\",\"418\"
,\"431\",\"437\",\"438\",\"450\",\"506\",\"514\",\"519\",\"548\",\"579\",\"581\",\"587
\",\"604\",\"613\",\"639\",\"647\",\"705\",\"709\",\"778\",\"780\",\"782\",\"807\",\"8
19\",\"825\",\"867\",\"873\",\"902\",\"905\"],\"node\":{\"type\":\"action\",\"values\"
:[{\"action_type\": \"set_carrier\",\"action_value\": \"0247\"},{\"action_type\": \"se
t_network_management_class\",\"action_value\": \"14\"}]}},{\"values\": [\"others\"],\"
node\":{\"type\":\"action\",\"values\":[{\"action_type\": \"set_carrier\",\"action_val
ue\": \"0222\"},{\"action_type\": \"set_network_management_class\",\"action_value\": \
"14\"}]}}]}}"
}
```


7. Error Detection

This document assumes that if the syntax of the HTTPS GET or POST is, itself, invalid, then the typical web server failures (e.g. HTTPS 404) occur; those failures are addressed outside of this specification. Once the HTTPS header is validated and the JSON syntax is parsed successfully, then the next level of error detection occurs.

The RouteLink Server uses an error field in the response to a request to communicate a parameter or JSON content failure. It also uses this response to communicate its inability to complete the request for any other reason. A message field is supplied to provide more information in free-form, printable, text (to aid client side debugging). The following is an example JSON response indicating that an error was encountered by the RouteLink Server:

```
{
      "error": "SeeTableBelowForPossibleValues",
      "message": "Explanatory text is included here."
}
```

The following table describes the possible values for the error field.

Error Field Values		
Error value	***Comments***	***Action***
"temporary"	A temporary error on the server has occurred. The HTTPS connection has been successful to this point, but the server is unable to complete the request at this time. Sending the same request again, but at a later time, is expected to be successful. An example could be a temporary database outage during maintenance.	The client SHOULD retry the request later, waiting at least 60 seconds before retrying.
"permanent"	A permanent error has occurred that MUST be addressed by the client. Resending the request will NOT yield success, so retrying is not reasonable without some additional action being taken first. An example is an invalid field in a request.	The client MUST stop sending the request and inspect the message field to learn more about the problem. Contacting Somos may be required based on the issue. Once the problem has been addressed, the client can resume communication.

If the client receives a response with an error value from the table above, it MUST take the action described in the table above.

The message field is free-form text, where the contents are otherwise not constrained by this specification. It allows the server to communicate the issue in printable text to aid client debugging. The client MUST NOT depend upon any values in the message text, since the text is subject to change at the discretion of the server at any time without notice. Up to 1000 printable characters (including whitespace such as carriage returns, etc.) are allowed for flexibility.

It is unnecessary (and would likely be incomplete) to try to enumerate all the possible errors; however, they all fall into one of the categories in the table above ("temporary" or "permanent") and are driven by the action the RouteLink Server must take. Below are some examples of errors to aid development. The text in these examples is NOT required in the message field -- it is only for explanatory purposes in this document.

Temporary error examples:

- A database outage has occurred at the RouteLink server.
- A maintenance window is in progress at the RouteLink server.

Permanent error examples:

- A field is missing in the request.
- An unknown field is present in the request.

8. Appendix

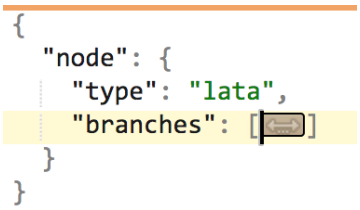
8.1 JSON Call Processing Record Structure

Call routing using the JSON Call Processing Record (CPR) is accomplished by traversing a hierarchy of nodes containing either decision-making information or actions to perform. These node types can be intermixed in any combination and any order. For example, action nodes may appear at the beginning, middle, or end of a given routing branch.

All values in the JSON CPR are treated as strings.

8.1.1 Node Containers

JSON node tags are the top-level container for multiple decision node types as well as action nodes. The below image shows a collapsed JSON call processing node. All nodes have an associated type as shows here.

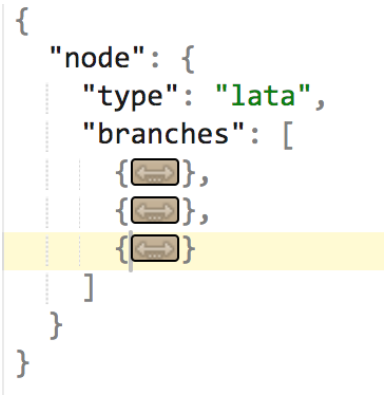


All decision trees start with a "node" identifier. The node "type" field indicates the criteria that must be matched by a given branch in order for that branch to be traversed further.

Node Tag	
Node Tag	***Description***
type	See the list of node types in the "Node Type" table below.
branches	An array of branches, each of which contains one or more values. The data contained in the values array depends on the branch type. More information is in the "Branches Tag" table below.
qualifiers	Tags • qualifier_type • qualifier_value See the "Qualifiers" table below for more information.
action_type	For "action" node types, this contains field contains the specific action to be performed in the event the values associated with the branch are a match to the routing criteria. More information can be found in the "Action Node" table below.
action_value	Value associated with the action type. Again, more information is in the "Action Node" table below.

8.1.2 Branch tag

The branches tag contains an array of value/node sets. The type of data included in the "values" tag depends on the node type with which the branch is associated.

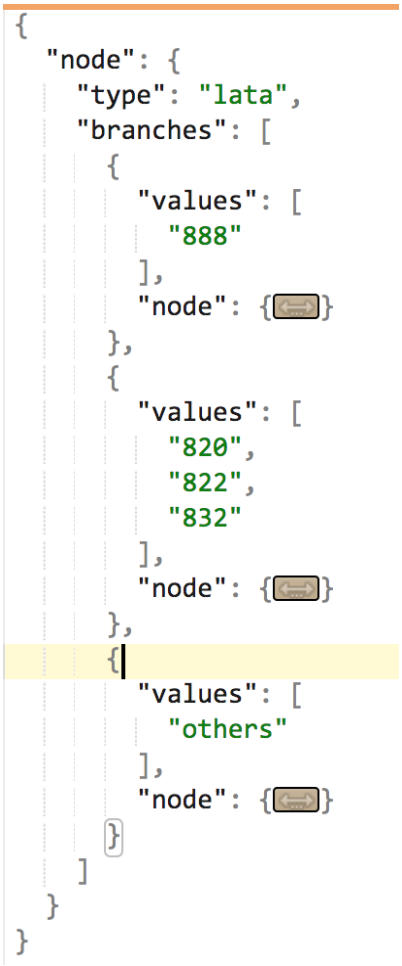


In the example above, there are three branches associated with the top-level node. Because the node type is "lata", which of the branches is traversed depends on the values associated with each.

Branches Tag	
Branches Tag	***Description***
values	Contains an array of one or more values whose type depends on the overall node type for the associated branch. Note that this field may contain either single values or a range. In the event a range is represented, the beginning and ending values are separated by a colon.
node	This field contains decision and/or action nodes that are traversed if any of the values are matched.

8.1.3 Decision Nodes Types

The table below contains all of the potential decision node tags and a brief description of each.



The figure above illustrates a single, top-level decision node fully expanded, exposing the relationship between the node, branch, and values tags.

Node Type	
Node Type	***Description***
day_of_week	1 -- Sunday 2 -- Monday 3 -- Tuesday 4 -- Wednesday 5 -- Thursday 6 -- Friday 7 -- Saturday
day_of_year	001 -- Jan 1 002 -- Jan 2 . . 059 – Feb 28 060 -- Feb 29 .

	. 366 -- Dec 31
lata	3-digit LATA (000-999)
npa	3-digit ANI (NPA)
npa_nxx	6-digit ANI (NPA-NXX)
npa_nxx_xxxx	10-digit ANI (NPA-NXX-XXXX)
nxx	3-digit ANI (NXX)
percent	
time_of_day	The time value(s) on the branches of the Time node are specified using the number of the quarter hour of the day. The time values range from 0 through 96. A value of "0" means 12:00AM if it is used at the beginning of the time range. A value of "96" means 12:00AM if it is used at the end of the time range. A value of "01" means 12:15AM. A value of "95" means 11:45PM.
others	This branch is taken when no other branch has matching values.
action	Tags: • action_type • action_value

NOTE: The time range in the time node is inclusive of the range starting value and exclusive of the range ending value. For example, the time range of 8:00am - 5:00pm means from 8:00:00am to 4:59:59pm. The range for day & date nodes is inclusive of the start and end values. For example MON-FRI means from 00:00 Monday to 23:59:59 Friday.

8.1.4 Action Nodes

The following table contains all of the potential action node tags and a brief description of each.

```
{
  "node": {
    "type": "lata",
    "branches": [
      {
        "values": [
          "820",
          "871",
          "884",
          "888"
        ],
        "node": {
          "type": "action",
          "values": [
            {
              "action_type": "set_final_treatment",
              "action_value": "out_of_band_announcement"
            }
          ]
        }
      },
      {
        "values": [
```

Shown here is a portion of a call processing record fully expanded to illustrate the decision-making process from the top-level to the actions. In this case, a call received from a LATA identifier contained in the "values" field (820, 871, 884, 888) would result in following the next level node associated with this branch. In this case, the next-level node would be the "action" node. In the case of the "action" node, the values are no longer routing criteria. Rather, this field contains one or more actions to be taken given this route resolution. The example above shows that the route here would result in returning a set final treatment action to play an out-of-band announcement would be returned to the calling entity.

Action Type	
Action Type	***Action Value***
set_carrier	Carrier code
set_final_treatment	Values: • out_of_band_announcement • vacant_code_announcement • disconnected_number_announcement • final_treatment_error
set_local_service_office	The LSO is represented by NPA-NXX
set_network_management_class	Indicates the network management class threshold
set_routing	10-digit routing number
set_template	10-digit template identifier. The final call routing information is obtained by fetching the routing structure associated with the template digits.

8.1.5 Qualifiers

Node qualifiers indicate time zone and whether or not daylight savings is in effect. Qualifier values are present only for date/time-related decision nodes. These nodes include time_of_day, day_of_week, and day_of_year.

Qualifiers	
Qualifiers	***Description***
qualifier_type	Values: • time_zone • dst_flag
qualifier_value	time_zone: 0 = NewFoundland Time 1 = Atlantic Time 2 = Eastern Time 3 = Central Time 4 = Mountain Time 5 = Pacific Time 6 = Yukon Time 7 = Hawaiian & Alaskan Time 8 = Bering Time dst_flag: 1 = DLS in not in effect 2 = DLS in effect

8.1.6 JSON CPR Examples

The following sections contain samples of JSON Call Processing Records.

Time of Day with Qualifiers

The example below demonstrates the structure of a JSON CPR containing a Time of Day node including branch qualifiers. It also shows the usage of the 15-minute declinations in determining the time of day range.

```
{
  "node": {
    "type": "lata",
    "branches": [
      {
        "values": [
          "820",
```

```
      "822",
      "824",
      "826",
      "828",
      "830",
      "870",
      "871",
      "884",
      "888"
    ],
    "node": {
      "type": "action",
      "values": [
        {
          "action_type": "set_final_treatment",
          "action_value": "out_of_band_announcement"
        }
      ]
    }
  },
  {
    "values": [
      "others"
    ],
    "node": {
      "type": "time_of_day",
      "qualifiers": [
        {
          "qualifier_type": "time_zone",
          "qualifier_value": "central_time_zone"
        },
        {
          "qualifier_type": "dst_flag",
          "qualifier_value": "dst_in_effect"
        }
      ],
      "branches": [
        {
          "values": [
            "0:48"
          ],
          "node": {
            "type": "action",
            "values": [
              {
                "action_type": "set_final_treatment",
                "action_value": "vacant_code_announcement"
              }
            ]
```

```
            }
          },
          {
            "values": [
              "48:96"
            ],
            "node": {
              "type": "action",
              "values": [
                {
                  "action_type": "set_final_treatment",
                  "action_value": "out_of_band_announcement"
                }
              ]
            }
          }
        ]
      }
    }
  ]
 }
}
```

Percent

This section contains a JSON CPR with a Percent decision node. Of particular note in decision nodes of this type is the absence of the "Others" branch as the final node of the decision tree. Percent decision nodes must contain branches for all percentage values. "Others" branches are not allowed.

```
{
  "node": {
    "type": "lata",
    "branches": [
      {
        "values": [
          "888"
        ],
        "node": {
          "type": "action",
          "values": [
            {
              "action_type": "set_carrier",
              "action_value": "0539"
            },
            {
              "action_type": "set_network_management_class",
```

```
          "action_value": "14"
        }
      ]
    }
  },
  {
    "values": [
      "820",
      "822",
      "832"
    ],
    "node": {
      "type": "action",
      "values": [
        {
          "action_type": "set_carrier",
          "action_value": "0222"
        },
        {
          "action_type": "set_network_management_class",
          "action_value": "14"
        }
      ]
    }
  },
  {
    "values": [
      "others"
    ],
    "node": {
      "type": "percent",
      "branches": [
        {
          "values": [
            "80"
          ],
          "node": {
            "type": "action",
            "values": [
              {
                "action_type": "set_carrier",
                "action_value": "0444"
              },
              {
                "action_type": "set_network_management_class",
                "action_value": "14"
              }
            ]
          }
```

```
            },
            {
              "values": [
                "20"
              ],
              "node": {
                "type": "action",
                "values": [
                  {
                    "action_type": "set_carrier",
                    "action_value": "0288"
                  },
                  {
                    "action_type": "set_network_management_class",
                    "action_value": "14"
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}
```