Notifications Service

Web API Integration

Date: 27/07/2022

Version: 5.0

Department: ECSD

Unclassified

Malta Information Technology Agency, Gattard House, National Road, Blata l-Bajda HMR 9010 Malta

Telephone: (+356) 21234710

Web Site: www.mita.gov.mt

Document Control Information

01. Document reference

V5.0

02. Document type

Documentation

03. Security classification

Unclassified

04. Synopsis

The purpose of this document is to outline a technical solution for the Notifications Platform WEB API to allow developers to programmatically access the Notifications Platform using a REST interface. In order to cover the required functionality, this document focuses on describing the solution that is used to support backend operations for the successful delivery of messaging data to various channels.

05. Document control

Author	Change API	Distribution controller
MITA	MITA	

This document may be viewed and/or downloaded from the IMS On-Line, which maintains the latest issues of all documents and forms.

06. Authorisation

Issuing authority	**Approval authority**
Signature / Date MITA	**Signature / Date** MITA

07. Modification history

Version	Date	Author	Comments
Version 1.0	20/03/2017	MITA	First version for release
Version 1.1	14/07/2017	MITA	Added new “Message Pricing” API.
Version 1.2	10/08/2018	MITA	Changed Accepted message Status to Pending to avoid confusion with SMSC Accepted message status. A new Accepted messages status with code 112 has been added to the existing Message Status list.
Version 2.0	12/02/2019	MITA	Change of title and logos
Version 3.0	01/10/2020	MITA	Updates to MITA Logo.
Version 4.0	30/07/2021	MITA	Added more details on how to generate the HMAC signature and header fields.
Version 5.0	27/07/2022	MITA	Added details on how to generate the hash for encoded body to generate the HMAC signature.

Table of Contents

1 Introduction

The Notifications API allows developers to programmatically access the various components within the Notifications Platform using a uniform REST interface. The Notifications API does not use the conventional username and password to provide access to client applications; an API Key and Secret are provided instead. A username/password pair can be used across multiple applications, are easier to intercept, and can be compromised. On the other hand, the API Key mechanism uses a 32-character Secret Key, which is securely randomly generated by the Notifications Platform, which provided significantly greater entropy, and thus makes it harder to compromise. Since Secret Keys are regenerated frequently (every seven days by default), they are much more challenging for potential attackers when compared to a normal eight-character password. Moreover, Secret keys provide limited exposure since the keys are never exposed in the UI or during a transaction.

To be able to integrate the Notifications API, developers are provided with an API account (API Key and Secret Key) issued by MITA from the Administration application on behalf of the government entity that they are administering. The Notifications API is made available over SSL, hosted on a server provided by MITA. Moreover, the Notifications API takes full advantage of HTTP headers, such as the Authorization header attribute, to authenticate and authorize an application consuming the API. Both HTTP request and response objects are developed in such a way to be meaningful as possible by making use of HTTP verbs (GET, POST, PUT, DELETE, etc.), individual headers (Location, Content-Type, Accept, etc.), and HTTP response status codes (200, 400, 404, etc.). Developers conducting the development and integration of the Notifications API require a strong working knowledge of the various HTTP components and general understanding of REST Web APIs.

2 Authentication

The Notifications API uses a variation of the OAuth 2 access authentication scheme[1], an HTTP authentication method using a Hash-based Message Authentication Code (HMAC) algorithm to provide cryptographic verification of portions of HTTP requests. A request to perform an action is accompanied by the correct Authorization header as per this specification.

HMAC is used to calculate a message authentication code using a hash function in combination with a shared secret key between the client application and the Notifications API. The main use of HMAC is to verify the integrity, authenticity, and the identity of the message sender. An API Key and Secret Key are generated by MITA administration from the Notifications Platform Administration application and are used by the client to create a unique HMAC (hash) representing the request originated from it to the server. This process happens only the first time when the client registers with the server. While the API Key remains the same, the client application is forced by the Notifications API backend to request a new Secret Key. The Secret Key remains valid for seven days (configurable) i.e. if the Secret Key expires, the client application requests a new Secret Key. The Secret Key is used to produce a unique hash derived from the combination of the request data (API Key, request URI, request content, HTTP method, timestamp, and nonce). Then, the client application sends the hash to the server, along with all information it was going to send already in the request.

Once the server receives the request along with the hash from the client application, it tries to reconstruct the hash by using the received request data from the client using both API Key and Secret Key. Once the hash is generated on the server, the server is responsible for comparing the hash sent by the client with the regenerated one; if these match, then the server considers the submitted request as authentic and can process it.

2.1 HMAC Client Authentication

The first time the client application connects to the Notifications API, they use a Secret Key (along with the API Key) provided by MITA administration. The Notifications API forces the client application to request a new Secret Key to be able to invoke any of the actions available. The client is responsible for securely storing the API Key and Secret Key, and never sharing it with other parties. The Secret Key remains valid for seven days (configurable); past this, the client application has to request a new Secret Key to perform subsequent actions. Prior initiating a request, the client application needs to compute a hash for the payload to send.

[1] https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-http-mac-01

```
Authorization: SMG-V1-HMAC-SHA256 id="{API Key}",
ts="{timestamp}", nonce="{unique random string}",
mac="{base64 encoded signature}"
```

Figure 2-1 Authorization Header

To calculate the required hash, the client application needs to build a string of value pairs by combining the following data elements: API Key, HTTP method (verb), request URI, request timestamp, nonce, and a base64 string representation of the request payload. Each data element is separated by the newline character ("\n"). The newline character is introduced (instead of a comma delimiter) between the value pairs to avoid the possibility to modify authentication queries (through a MITM and replay attacks) by moving parts of the data from one variable to another. This string is hashed using HMAC SHA256 hashing algorithm with the assigned Secrete Key, resulting in a unique signature for this request. The signature is sent in the Authorization header using a custom scheme called "SMG-V1-HMAC-SHA256". Note that the version number in the scheme name will allow for different future authentication implementations without breaking the current authentication contract mechanism between client and server applications. The client then sends the request payload along with the data generated in the Authorization header as shown in Figure 2-1. The value of the header is made up of the components shown in Table 2-1.

Table 2-1 Authorization Header Fields

Field	Description
id	Your API Key, issued by MITA. To generate a new API Key, log to the Notifications Portal, go to API Key Manager and click on the "Create New" button. Details on managing API Keys can be found in the Notifications System User Manual.
ts	UNIX timestamp of the date and time the request was made. The request timestamp is calculated using UNIX time (number of seconds since 1st January 1970) to overcome any issues related to a different time zone between client and server. To avoid sync issues, a slight buffer is allowed.
nonce	A randomly generate string used to ensure that each request is unique and to prevent replay attacks. **Make sure the nonce does not exceed 36 characters**.
mac	This is the Base64 encoded hash of the request.

The **mac** hash is a SHA-256 digest of a series of concatenated strings related to the request called the signature string. The signature string is based on the following fields:

- API Key
- HTTP Request Method (e.g., GET, POST, PUT, DELETE)
- HTTP Request URI
- Timestamp
- Nonce
- Request content (JSON or XML String) SHA-256 digest Base64-encoded

Each part of the string is separated by a line feed character (\n; **not \r or \r\n**). Figure 2-2 shows an example for a POST request to the `/api/v1/messages endpoint`. The `\n` characters are shown for clarity. The newline character is introduced (instead of an inline delimiter) between the values to avoid the possibility to modify authentication queries (through a MITM and replay attacks) by moving parts of the data from one variable to another.

```
0123456789ABCDEF0123456789ABCDEF\n
POST\n
https%3a%2f%2fnotifications-api.gov.mt%2fapi%2fv1%2fmessages\n
1627656100\n
```

Figure 2-2 MAC Hash Signature Example

Once the signature string has been constructed, it must be hashed using the HMAC method, with the API secret (issued with your API key from the API Key Manager) used as the hash key. The Notifications System uses the SHA-256 algorithm for hashing. It is recommended that you use pre-existing libraries (e.g., `System.Security.Cryptography.HMACSHA256` in .NET, `javax.crypto.Mac` in Java, `hash_hmac` in PHP, `hmac` and `hashlib` in Python, etc.) to calculate the hash to avoid issues during validation at the server. **Ensure the hash is output in binary, and not in hexadecimal.** Once the hash is calculated, Base64 encode it and include it in the HTTP header. An example of the hashed string can be seen in Figure 2-3.

```
wjm7prVUD/n3w3nk+C0EWwod98oz4O9T+z+Wy1rIXQ0=
```

Figure 2-3 Base64 Encoded HMAC Hash

All requests to the Notifications API endpoints must be accompanied by an Authorization header as per this specification, as shown in Figure 2-1. The server receives the payload included in the request, along with the Authorization header. The server looks at the header parameters, computes the same HMAC hash and validates the key using the same secret. If you receive a 401 unauthorised error, the following is a list of things you need to check:

- Make sure the API Key is not expired and the renewed key was successfully stored by your application. This can be checked by logging to the Notifications Portal and go to the API Key Manager.
- It is important that each application has its own API Key. Since API Keys expire after seven days, one application may invalidate the old key and generate a new one. Applications using the old key will be denied access.
- Make sure that the time you are authenticating with is correct and in UTC. The server must have accurate time as the authentication MAC is building using this time. If the time is not in sync with the Notifications API server or not using UTC, authentication will fail.
- Since the client and server generate the hash (signature) using the same hashing algorithm, parameter order must be observed as per specifications here. Any slight change, including

case sensitivity, when generating the signature will result in a totally different signature and all requests from the client to the server will be rejected.

- It is important that the same nonce from your server is not sent more than once. The nonce field must be unique with every request. More over the nonce must not exceed 36 characters in length.

2.2 API Secret Key Expiration

To mitigate any security risks, as explained in the previous section, the Notifications System does not generate long-lived Secret Keys. The shared Secret Key between the client application and the Notifications API needs to be changed from time to time. By default, the Secret Key remains valid for a period of seven days (maximum three months) and any API call performed using an expired key will result in a `HTTP Error 205 Reset Content,` with a message saying that the Secret Key is expired. To refresh the API Key, you can do this from either the Notifications Portal -> API Key Manager, where the key needs to be updated manually, or by invoking the “GET api/v1/key” API endpoint. It is suggested that latter method is to be implemented. See section 6.7.1 for more details on how to refresh the API Key.

3 Notifications API Response Codes

The Notifications API makes use of HTTP standards, specifically, verbs and status codes. Verbs allow for API calls to have an intent, i.e. to get some information (GET), post new data (POST), update existing data (PUT), delete existing data (DELETE), etc. HTTP status codes on the other hand provide a way to respond to the Notifications API calls without having to develop custom schemes. The following response codes, shown in Table 3-1 apply to all requests.

Table 3-1 Notifications API Status Codes (as per HTTP/1.1 RFC 2616)

Status Code	Meaning	Description
200	OK	The request was processed successfully by the Notifications Platform and an entity describing or containing the result of the action was returned.
201	Created	The request was fulfilled and a new resource (e.g. message, service, subscription, etc.) was created.
202	Accepted	The request has been accepted for processing, but the processing has not been completed. This is mainly used when a new message is submitted.
204	No Content	The request was processed successfully by the Notifications Platform, but the response was intentionally left empty.
205	Reset Content	Authentication failed because the API Key has expired. The request should be sent to the Notifications Platform again with a renewed API Key.
400	Bad Request	The request contains invalid or missing data. A list of validation errors is returned with the response.
401	Unauthorized	Authentication failed or the `Authorization` header was not provided or incorrect HMAC signature.
402	Payment Required	The request is accessible, but reserved for future logic that is not yet in place.
403	Forbidden	Authentication failed because the API Key is either inactive or expired. If API Key is expired, the client application is required to request a new Secret Key using the current Secret Key and API Key.
404	Not Found	The URI does not match any of the recognised resources, or, if a specific resource with does not exist.
405	Method Not Allowed	The HTTP request method is not allowed.
406	Not Acceptable	The `Accept` content type is not supported by the Web API.
410	Gone	The request is no longer available.
415	Unsupported Media Type	The `Content-Type` header is not supported by the Web API.
500	Internal Server Error	The request encountered an internal issue and an error was thrown. The processing may have partially been performed.

4 Notifications API Media Types

The Notifications API uses either XML or JSON to encode data sent in requests or received in responses. The media type of the data returned are controlled by setting the appropriate HTTP Headers for each request. Since some languages may have better libraries for parsing data in JSON format, as opposed to XML, the developer integrating the API I may decide to set the media type to JSON for all requests.

When submitting a request, the `Content-Type` header is used to specify the format the data is in. For JSON, `application/json` is used, while for XML, `application/xml` is used. To specify the output desired format, either in XML or JSON, the `Accept` header is used. If the `Accept` header cannot be used, one can use the format parameter in the query string. The format parameter takes precedence over the `Accept` header.

5 Other Design Considerations

5.1 Versioning

The Notifications API makes use of best practices to better address future requirements and effectively scale the API without disrupting the existing services being offered. New requirements will be addressed more gracefully without the need to create a new integration from scratch, allowing for a more stable and scalable architecture. To address new future requirements each endpoint of the API is versioned accordingly. This has the benefit of providing a more stable contract between the client application and the server. Client applications making use of an older version of the Notifications API will not break, allowing for the Notifications API evolving with new future requirements.

```
Example 1: POST /api/v1/messages/create
Example 2: POST /api/v2/messages/create
```

Figure 5-1 Notifications API Versioned Endpoint

Figure 5-1 shows how versioning is used to address a new change for the Notifications API "messages" endpoint. The URI itself is used to identify a different resource, denoted by `v{n}`, where `{n}` is the version number. While retaining the context, Example 1 may implement a completely different functionality from Example 2. One benefit to this approach is that it makes sure that the version isn't optional and has to be included in the URI. As a result, the client application is only required to change only when moving to a new version.

5.2 Pagination

To allow for faster retrieval of messages from the server, the Notifications API implements pagination. This means that instead of returning hundreds or thousands of records at once, the API will limit the results in the response to a number as specified by the client application in the URL. The client application will use two query string parameters, `PageIndex` and `PageSize` to paginate as desired. The `PageIndex` parameter is a number used to choose which page is required, while the `PageSize` parameter is used to limit the number of records returned from the page index. For example, if a dataset contains 20 items and the client requires 10 items per page, the `PageIndex` parameter is set to 1 for the first page and 2 for the second page, with the `PageSize` parameter set to 10. An example is shown in Figure 5-2 where a request is done for the list of subscribers for the service with Id set to 45.

```
GET /api/v1/services/45/subscribers?PageIndex=2&PageSize=10
```

Figure 5-2 Notifications API Pagination

6 Notifications Platform REST Notifications API Interface

The Notifications API is hosted on a server that listens for HTTP requests containing message objects encoded in JSON or XML. The API process the HTTP requests message data and sends the data to the appropriate channel for onward delivery. The API interface sends back a JSON response containing information about the submitted message/s such as the message identifiers and status of the sent message/s. The interface also includes a callback function which sends back delivery reports. This callback function achieves this by sending an HTTP Request to a public HTTP endpoint hosted on the client side, capable of parsing the delivery reports. Figure 6-1 and Figure 6-2 illustrate the high-level architectural setup for both processes.

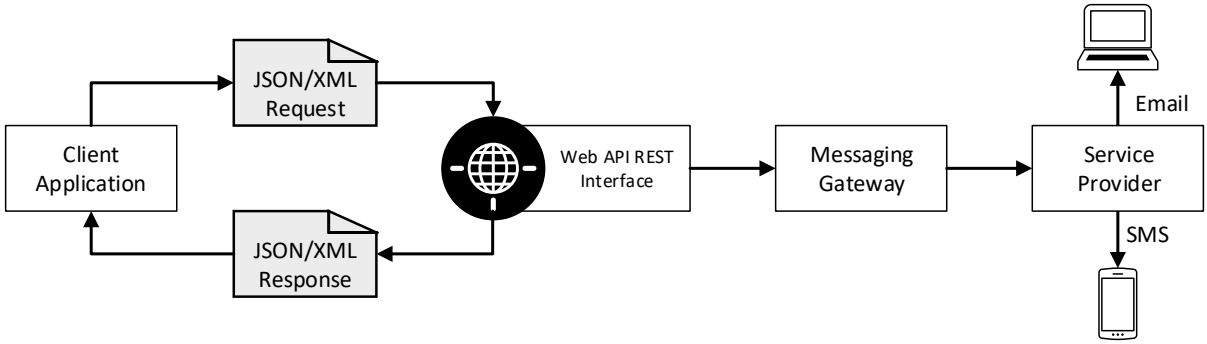

Figure 6-1 Message Rest Web API Call

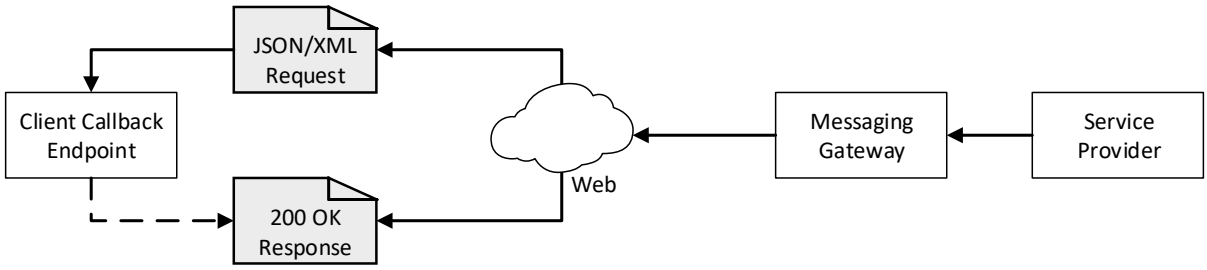

Figure 6-2 Message Callback Acknowledging Message Status

The following section further discusses in detail the REST interface, endpoints and resources available via the API. The allowed request methods are explained for each resource in terms of the service contract interface and the exchange of data objects that the both the API and client application are processing.

6.1 Message API

The Message API interface is responsible for handling incoming HTTP requests containing message data in order to send the required message for onward delivery. The Message API is responsible for sending back an appropriate HTTP response (depending on the message type) containing information about the submitted message such as message identifiers generated by the Notifications Platform. Table 6-1 lists the allowed request methods for the Message API.

Table 6-1 Message API REST API Endpoints

API	Description
`GET api/v1/messages/{messageId}`	Returns a Message object containing details and status.
`POST api/v1/messages`	Creates a new Message request object for onward delivery.
`PUT api/v1/messages/{messageId}`	Updates a Message object stored in the Notifications Platform operational repository scheduled to be sent at a later date.
`DELETE api/v1/messages/{messageId}`	Deletes a Message object stored in the Notifications Platform operational repository.
`PUT api/v1/messages/{messageId}/schedules`	Updates a Message object stored in the Notifications Platform that is awaiting delivery with a new Scheduled Delivery Date. If the Message object is enroute to the service provider, no updates will be made.

6.1.1 [GET] Get Messages

Though message delivery status updates are delivered through the callback function, the client application is able to submit a request for this using the HTTP GET request method over the "`api/v1/messages/{messageId}`" REST API endpoint. This API call looks up a single message object as specified by its unique identifier in `{messageId}` placeholder and returns an abstraction of the `Message` object, containing both the message details sent and the current delivery status. Table 6-2 shows the `DeliveryReport` object that is returned by the API call.

6.1.1.1 Response

This request method returns a status code 200 OK with a collection of `DeliveryReport`. A status code 404 Not Found is provided if no messages are found.

Table 6-2 `DeliveryReport` Object

Name	Description	Type
`MessageId`	Notifications Platform unique numeric identifier.	`globally unique identifier`
`BatchId`	Unique reference generated by the Notifications Platform to identify a group of messages posted in batch. This is useful in situations where a group of messages needs to be updated or deleted.	`globally unique identifier`
`Contact`	The Contact property contains the recipient's delivery address where the message was sent to.	`Contact`
`Language`	The Language property specifies which language this `MessageContent` is for.	`Language`
`Subject`	This is only required for Email message types or other message types requiring a subject.	`string`
`MessageBody`	Depending on the message type, this fields allows for Unicode characters and a limit to the number of characters that can be sent.	`string`
`Attachments`	This represents a collection of attachments to be sent along with the message body. This property can only be used for Email message types or other message types allowing an attachment.	`Collection of AttachmentUri`
`MessageStatus`	This property is used to determine the current status of the message. See Table 6-7 for a complete list of statuses.	`MessageStatus Enum`
`DateCreated`	Date when the message was submitted.	`date (ISO-8601)`
`DateUpdated`	Date when the message was last updated.	`date (ISO-8601)`
`ClientReference`	Unique reference generated by the client application.	`string`
`MessageType`	Message Type e.g. Email or SMS	`MessageType Enum`
`MessagePriority`	Message Priority – Low, Normal, and High. These are used by the Message Broker to prioritize message delivery.	`MessagePriority Enum`
`SenderId`	Sender Identifier used by the Notifications Platform to specify the sender name for the message.	`globally unique identifier`
`CallbackURL`	A URL provided during Message creation.	`string`
`ScheduledDelivery Date`	Date and time in ISO-8601 format for the message to be scheduled for delivery.	`date (ISO-8601)`

Table 6-3 `Contact` Object

Name	Description	Type	Required
`DisplayName`	This is the Display Name for the recipient and is optionally used for Email message types.	`string`	Optional
`Title`	Salutation e.g. Mr., Ms., Dr., etc.	`Salutation` Enum	Optional
`FirstName`	Recipient's first name.	`string`	Optional
`LastName`	Recipient's last name.	`string`	Optional
`Email`	If Email message type is selected this field is required.	`string`	Conditional
`MobileNo`	This is the recipient's MSISDN and abides to international formatting where only digits between 7-15 digits are allowed. This field is only required if SMS message type is selected.	`string`	Conditional

Table 6-4 `Salutation` Enumeration

Name	Value	Description
`Mr`	Mr	
`Miss`	Miss	
`Mrs`	Mrs	
`Ms`	Ms	
`Dr`	Dr	
`Prof`	Prof	

Table 6-5 `Language` Enumeration

Name	Value	Description
`English`	en	
`Italian`	it	
`German`	de	
`French`	fr	
`Spanish`	es	
`Maltese`	mt	

Table 6-6 `AttachmentUri` Object

Name	Description	Type
`Uri`	Uri to download attachment.	`string`
`Size`	File size.	`Integer`
`MD5`	File checksum.	`string`
`FileName`	Name of the file.	`string`
`ContentType`	MIME type associated with file.	`string`

Table 6-7 `MessageStatus` Enumeration

Name	Value	Description
`Accepted`	100	The message has been processed by the Notifications Platform and awaiting queuing. Scheduled messages will also show this status.
`Sent`	105	The message has been pushed to the Message Broker queue and is awaiting routing to the appropriate operator carrier.
`Enroute`	110	The message is currently being delivered and awaiting acknowledgment from the operator.
`Accepted`	112	The message has been accepted and processed by the operator and awaiting delivery to the designated recipient.
`Delivered`	115	The message has been delivered and received acknowledgment from the operator that the message was received by the recipient.
`Undelivered`	120	The message was not delivered to the designated recipient due to an upstream operator issue.
`Expired`	125	The message has exceeded the maximum number of attempts or time to live parameters.
`Failed`	130	The message was rejected by the Notifications Platform.
`InvalidAddress`	135	The supplied address that is being used to send the message was not valid.
`Rejected`	140	The message was rejected by the operator.
`Unknown`	145	The message was rejected for an unknown reason by the operator.
`SystemError`	150	The message was not delivered due to a Notifications Platform error.
`Acknowledged`	160	The consuming application was notified that the message was processed.
`NoConnection`	170	The message cannot be delivered at the moment due to a connection error with the Service Operator server.
`MessageQueueFull`	180	The Service Operator has too many queued messages and temporarily cannot accept any more messages or server has too many messages pending for the specified recipient and will not accept any more messages for this recipient until it is able to deliver messages that are already in the queue to this recipient.

Table 6-8 `MessageType` Enumeration

Name	Value	Description
`Email`	email	Marks the message as an Email.
`Sms`	sms	Marks the message as an SMS.

Table 6-9 `MessagePriority` Enumeration

Name	Value	Description
`Normal`	100	Gives normal priority to message during dispatch.
`High`	200	Gives higher priority to message during dispatch.

6.1.2 [POST] Post a New Message

Messages are sent by the client application using an HTTP POST request method through the "`api/v1/messages`" REST API endpoint. This endpoint offers a single API reference to send messages of various types (for Phase 1 either SMS or Email) and can be used by the client application to send a `Message` object to either an individual recipient or multiple. This API allows only for one message to multiple recipients per request to be submitted, however a separate API interface is available to send messages of different types to multiple recipients in batch (to be discussed in the following sections). Table 6-10,

Table 6-11, and Table 6-12 represent the level within the Request object, starting with the outermost one.

The client application creates a `Message` object and submits this using the HTTP POST request method. On success, the response will contain a `Location` header that provides the URI for the new message object with its unique identifier. Since, the Message API allows for multiple destinations, the URI will contain the `BatchId`, not the `MessageId`. The `BatchId` can then be used to retrieve the status of individual messages at a later stage. If invalid data is provided (status code 400 Bad Request), the response body contains the Model State array listing the errors for each field. Note, that the response status code is not the actual delivery status of the message. Information about the delivery status can be retrieved from the `DeliveryReport` object (discussed in the following sections).

6.1.2.1 Request Body

This request method returns a status code 200 OK with a collection of `DeliveryReport`. A status code 404 Not Found is provided if no messages are found.

Table 6-10 `Message` Object

Name	Description	Type	Required
`Contacts`	The Contacts property is a collection of subscribers to specify the delivery address where the message is sent. Depending on the message type, this can be either an MSISDN or Email address.	`Collection of Contact`	Required
`MessageContent`	A collection of `MessageContent` that defines the content of the messages to be delivered.	`Collection of MessageContent`	Required
`ClientReference`	Unique reference generated by the client application, which are echoed back to the client in the callback, along with the message details.	`string`	Required
`MessageType`	Message Type e.g. Email or SMS	`MessageType Enum`	Required
`MessagePriority`	Message Priority – Low, Normal, and High. These are used by the Message Broker to prioritize message delivery.	`MessagePriority Enum`	Required
`SenderId`	Sender Identifier used by the Notifications Platform to specify the sender name for the message.	`globally unique identifier`	Required
`CallbackURL`	If provided, the notifications platform will deliver a receipt back to the application through the HTTP POST method for the provided URL. If URL is not provided the default URL are taken from the API Key Account information.	`string`	Optional
`ScheduledDelivery Date`	Date and time in ISO-8601 format for the message to be scheduled for delivery.	`date (ISO-8601)`	Optional

Table 6-11 `MessageContent` Object

Name	Description	Type	Required
`Language`	The Language property specifies which language this `MessageContent` is for.	`Language Enum`	Required
`Subject`	This is only required for Email message types or other message types requiring a subject.	`string`	Conditional
`Body`	Depending on the message type, this fields allows for Unicode characters and a limit to the number of characters that can be sent.	`string`	Required
`Attachments`	This represents a collection of attachments to be sent along with the message body. This property can only be used for Email message types or other message types allowing an attachment.	`Collection of Attachment`	Optional

Table 6-12 `Attachment` Object

Name	Description	Type	Required
`ContentStream`	The file content stream for this attachment.	`Collection of byte`	Required
`FileName`	File name.	`string`	Required
`ContentType`	Content type (MIME) associated with the file attachment.	`string`	Required

6.1.2.2 Response

This request method returns a status code 200 OK with a collection of `DeliveryReport`. A status code 404 Not Found is provided if no messages are found.

On success, the response will contain a `Location` header that provides the URI for the updated message object. If invalid data is provided (status code 400 Bad Request), the response body will contain the Model State array, listing the errors for each field. Otherwise, a MessageBatch object is returned (Refer to Table 6-13 `MessageBatch` Object). If the daily quota in the `Sender` account has been exceeded, a status code 402 Payment Required is sent.

Table 6-13 `MessageBatch` Object

Name	Description	Type
`BatchId`	Unique reference generated by the Notifications Platform to identify a group of messages posted in batch.	`globally unique identifier`

6.1.3 [PUT] Update Message

If a message has to be posted and is awaiting delivery, due to a future scheduled delivery date, the client application is able to update any of the details submitted. Messages are updated using the HTTP PUT request method through the "`api/v1/messages/{messageId}`" REST API endpoint, where the {messageId} placeholder is the unique identifier of the message to be updated. Note that the API call will only update a single message, single recipient. Batch updates are performed by the Batch Message API, discussed in the following sections.

6.1.3.1 Request Body

Table 6-14 `MessageUpdate` Object

Name	Description	Type	Required
`Contact`	The Contact property contains the recipient's delivery address where the message was sent to.	`Contact`	Required
`Subject`	This is only required for Email message types or other message types requiring a subject.	`string`	Conditional
`MessageBody`	Depending on the message type, this fields allows for Unicode characters and a limit to the number of characters that can be sent.	`string`	Required
`Attachments`	This represents a collection of attachments to be sent along with the message body. This property can only be used for Email message types or other message types allowing an attachment.	Collection of `Attachment`	Optional
`ClientReference`	Unique reference generated by the client application, which are echoed back to the client in the callback, along with the message details.	`string`	Required
`MessageType`	Message Type e.g. Email or SMS	`MessageType` Enum	Required
`MessagePriority`	Message Priority – Low, Normal, and High. These are used by the Message Broker to prioritize message delivery.	`MessagePriority` Enum	Required
`SenderId`	Sender Identifier used by the Notifications Platform to specify the sender name for the message.	`globally unique identifier`	Required
`CallbackURL`	If provided, the notifications platform will deliver a receipt back to the application through the HTTP POST method for the provided URL. If URL is not provided the default URL are taken from the API Key Account information.	`string`	Optional
`ScheduledDeliveryDate`	Date and time in ISO-8601 format for the message to be scheduled for delivery.	`date` (ISO-8601)	Optional

6.1.3.2 Response

On success, a status code 202 Accepted will contain a `MessageBatch` object (refer to Table 6-13 `MessageBatch` Object) as well as a `Location` header, which provides the URI for the updated message object. If invalid data is provided (status code 400 Bad Request), the response body will contain the Model State array, listing the errors for each field.

6.1.4 [DELETE] Delete Message

Similar to the update message API endpoint described in the previous section, if the message has not been delivered yet by the Notifications Platform the client application is able to stop delivery by deleting

the message. Messages are able to be deleted using the HTTP DELETE request method through the "`api/v1/messages/{messageId}`" REST API endpoint, where the {messageId} placeholder is the unique identifier of the message to be deleted. The client application will only be able to delete one message at a time using the API endpoint. If a batch of messages needs to be deleted this is done by the Batch Message API.

6.1.4.1 Response

The delete request returns the status code 204 No Content if it is successful. In case the message is already enroute, a status code 400 Bad Request is sent instead. If an invalid `MessageId` was provided, then a status code 404 Not Found is returned. Missing, or already deleted, messages return a status code 410 Gone.

6.1.5 [PUT] Change Message Scheduled Delivery Date

After message creation, the scheduled delivery date of the message can still be easily modified while it is still pending delivery. The date is modified using the HTTP PUT request method through the "`api/v1/messages/{messageId}`" REST API endpoint, where the {messageId} placeholder is the unique identifier of the message whose date is to be changed. The client application will only be able to modify one message delivery date at a time using the API endpoint. If a batch of message dates need to be updated, this is done by the Schedule API.

6.1.5.1 Request Body

Table 6-15 `MessageSchedule` Object

Name	Description	Type	Required
`ScheduledDeliveryDate`	Date and time in ISO-8601 format for the message to be scheduled for delivery.	`date` (ISO-8601)	Required

6.1.5.2 Response

The update request returns the status code 202 Accepted if it is successful, alongside a `MessageBatch` object (refer to Table 6-13 `MessageBatch` Object) and a `Location` URI, to retrieve and query the message. In case of invalid data, a status code 400 Bad Request is sent instead. If an invalid MessageId was provided, then a status code 404 Not Found is returned.

6.1.6 Push Status Notifications (Callback)

The push status notifications feature allows for the Notifications Platform to provide asynchronous updates on the delivery status of individual messages. The notifications are pushed using the Callback URI submitted with the message data or the default Callback URI defined in the API Key account details. The client application is able to specify the encoding of the request body (`DeliveryReport` object -

Table 6-2) by appending the `format` (value set to either XML or JSON) parameter at the end of callback URI. If the `format` parameter is not provided, by default the request body is encoded in JSON. A `DeliveryReport` object is submitted using HTTP POST request method and including the `Authorization` header similar to the one specified in section 2.1**Error! Reference source not found.**. An example of the `Authorization` header for the callback HTTP POST request is shown in Figure 6-3.

```
Authorization: SMG-V1-HMAC-SHA256 id={API Key}, ts={timestamp},
nonce={unique random string}, mac={base64 encoded signature}
```

Figure 6-3 Authorization Header for Callback Request

The push status notifications feature uses the API Secret Key to calculate a hash for the following data elements: API Key, HTTP method (verb), request URI, request timestamp, nonce, and a base64 string representation of the request payload. Each data element is separated by a newline character ("\n"). This string is hashed using HMAC SHA256 hashing algorithm with the assigned Secrete Key, resulting in a unique signature for this request. The signature is sent in the `Authorization` header using a custom scheme called "SMG-V1-HMAC-SHA256". The client application will receive that data included in the request along with the `Authorization` header and extracts the following data elements from the request: API Key, request timestamp, nonce and signature (MAC). The client application should use the current API Secret Key to generate the hash and compare the received hash in the `Authorization` header with the one generated locally. This method validates the authenticity of any message received from the Notifications Platform and thus avoiding attacks such as MITM and replay attacks.

Note that the Notifications Platform validates the callback URI to check if a callback can be completed. If the callback URI submitted along with the message data is invalid, a message is displayed indicating an invalid URI, and stops the delivery process of the message until it is fixed by the client. The Notifications Platform callback feature makes several attempts to deliver a callback notification to the specified callback URI. Each attempt is separated by an allowed time interval ranging from few minutes to hours. If the callback maximum number of attempts is exceeded (not more than three attempts), any further callback attempts are stopped. When a client receives a callback, the Notifications Platform expects to receive an HTTP 200 OK. The use of SSL at the client's callback receiver server is optional, however if HTTPS validation fails, the callback status is treated as failed.

6.2 Batch Message API

The Batch Message API interface is responsible for handling incoming batch message requests. This is similar to the Message API; however, the API can accept a collection of messages of different types. This will provide added flexibility from an administration perspective, e.g. administration can limit access

to batch functionality to individual organisations by assigning a different role which accepts only single message delivery.

From a client perspective, there can be scenarios where different types of messages to different recipients needs to be sent e.g. daily reminders where different subscribers have subscribed to different message mediums as their preference. Another scenario would be for personalised messages, where mail-merging text messages can be used to customize message content with minimal processing effort required from the client application.

Table 6-16 lists the allowed request methods for the Batch Message API.

Table 6-16 Batch Message API REST API Endpoints

API	Description
`GET api/v1/batches /{batchId} /messages`	Returns a paginated collection of `DeliveryReport` objects for a particular `BatchId` as specified by the `{batchId}` placeholder. The `DeliveryReport` object will contain both details and status for each message.
`POST api/v1/batches`	This endpoint allows a client application to send messages in batch of different types for onward delivery.
`PUT api/v1/batches /{batchId}`	Updates a batch of Message objects stored in the Notifications Platform operational repository scheduled to be sent at a later date.
`DELETE api/v1/batches /{batchId}`	Deletes a collection of Message objects stored in the Notifications Platform operational repository for the specified `BatchId`.

6.2.1 [GET] Batch Messages Status Lookup

Using the `BatchId` returned in the message creation response, the delivery status for each message with the batch can be looked up using an HTTP GET request method over the "`GET api/v1/batches/{batchId}`" REST API endpoint. This API call will look up the batch as specified by the `{batchId}` placeholder in the URI and returns a collection of `DeliveryReport` objects, containing both the message details and the delivery status for each message in the collection.
Additional parameters can be given in the URL, as shown in Figure 6-4 below. `PageIndex` denotes the desired page in the pagination (default 1 if not specified), while `PageSize` defines the collection size of each page (default 50 if not specified). `SortField` is used to anchor sorting on a specific field, and `SortDirection` defines the direction of sorting (Asc or Desc).

```
GET
api/v1/batches/{batchId}/messages?PageIndex={PageIndex}&PageSize={PageSize}&SortField={SortField}&SortDirection={SortDirection}
```

Figure 6-4 Batch Message Lookup Pagination

6.2.1.1 Response

Paginated responses return a status code 200 OK with both a `Page` object and the result, in the form of a `Collection` of objects. Refer to Table 6-17 Paginated Response and Table 6-18 `Page` Object below for the general structure. In case the `batchId` given is invalid, a status code 404 Not Found is provided instead.

Table 6-17 Paginated Response

Name	Description	Type
`Page`	Object containing pagination information.	`Page`
`Collection`	Collection of objects in this particular page (in this case, `DeliveryReport`).	`Collection`

Table 6-18 `Page` Object

Name	Description	Type
`Index`	Page index in this pagination sequence.	`integer`
`Size`	Page size for this pagination sequence.	`integer`
`Count`	Total amount of objects in the complete collection.	`integer`
`PreviousUri`	URI for previous page (i.e `Index-1`).	`string`
`NextUri`	URI for next page (i.e `Index+1`).	`string`

6.2.2 [POST] Post a New Batch Message Request

New batch message requests are sent by the client application using HTTP POST request method through the "`api/v1/batches`" REST API endpoint. The endpoint allows different message types grouped in a collection object (of type `Message`) to be compiled by the client application, and sent within a single request. The Notifications Platform will then parse the request and route the different message types to their respective delivery channel. On success, the response will contain a `Location` header that provides the URI for the new created messages grouped by their `BatchId`. The `BatchId` can then be used to retrieve the status of individual messages at a later stage. If invalid data is provided

(status code 400 Bad Request), the response body will contain the Model State array listing the errors for each field.

6.2.2.1 Request Body

Creating a batch of messages requires a collection of `Message` to be specified. Refer to Table 6-10 `Message` Object for the actual structure.

6.2.2.2 Response Body

Similar to individual message creation, this request returns a collection of `MessageBatch` (refer to Table 6-13 `MessageBatch` Object) objects, with each object being a reference to a `Message` in the same index in the request.

6.2.3 [PUT] Update Batch Messages

The update batch message functionality allows for a client application to update the data for a collection of messages awaiting delivery identified by their `BatchId`. Messages are updated using the HTTP PUT request method through the "`api/v1/batches/{batchId}`" REST API endpoint, where the `{batchId}` placeholder is the unique identifier to lookup the message to be updated for a particular batch. The client application will create a collection of `MessageBatchRequest` objects which are re-submitted using the HTTP PUT request method.

6.2.3.1 Request Body

Table 6-19 `MessageBatchRequest` Object

Name	Description	Type	Required
`MessageContent`	Collection of `MessageContent`	`Collection of MessageContent.` Refer to Table `6-11` MessageContent Object	Required
`ClientReference`	Unique reference generated by the client application, which are echoed back to the client in the callback, along with the message details.	`string`	Required
`MessageType`	Message Type e.g. Email or SMS	`MessageType` Enum	Required
`MessagePriority`	Message Priority – Low, Normal, and High. These are used by the Message Broker to prioritize message delivery.	`MessagePriority` Enum	Required
`SenderId`	Sender Identifier used by the Notifications Platform to specify the sender name for the message.	`globally unique identifier`	Required
`CallbackURL`	If provided, the notifications platform will deliver a receipt back to the application through the HTTP POST method for the provided URL. If URL is not provided the default URL are taken from the API Key Account information.	`string`	Optional
`ScheduledDelivery Date`	Date and time in ISO-8601 format for the message to be scheduled for delivery.	`date (ISO-8601)`	Optional

6.2.3.2 Response

On success, the response will be a status code 204 No Content, containing a `Location` header that provides the URI for the updated message object. If invalid data is provided (status code 400 Bad Request), the response body will contain the Model State array, listing the errors for each field. If an invalid BatchId was provided, then a status code 404 Not Found is returned.

6.2.4 [DELETE] Delete Batch Messages

Messages can be either deleted individually using the Message API delete API endpoint (see section 6.1.4) or using the BatchId to delete a collection of messages which was previously submitted for processing. This is only valid for undelivered scheduled messages which allows the client application to stop the delivery if required. The client application are able to delete batched messages using the

HTTP DELETE request method through the "`api/v1/batches/{batchId}`" REST API endpoint, where the `{batchId}` placeholder is the unique identifier for the group of messages to be deleted.

6.2.4.1 Response

On success, the response will be a status code 204 No Content. If an invalid BatchId was provided, or the batch is already deleted, a status code 410 Gone is returned. If any messages in the batches provided are already enroute, a 400 Bad Request is returned.

6.3 Message Schedule API

The Message Schedule API mainly allows for the retrieval of message schedule reports. This allows client applications to acquire a full report of messages to be dispatched in a given date. An additional request method also allows for the updating of scheduled delivery dates for multiple messages in a batch.

Table 6-20 Message Schedule API REST API Endpoints

API	Description
`GET api/v1/schedules`	Returns a collection of DeliveryReport for messages that are being dispatched today.
`GET api/v1/schedules/{year}/{month}/{day}`	Returns a collection of DeliveryReport for messages that are being dispatched in a specified date.
`PUT api/v1/schedules/{batchId}/batches`	Updates all the messages in a specific Batch with a given ScheduledDeliveryDate.

6.3.1 [GET] Today's Scheduled Messages

A message schedule list for today can be retrieved using the HTTP GET "`api/v1/schedules`" request method. Like Message Status Lookup (refer to Figure 6-4 Batch Message Lookup Pagination), additional parameters can be given to define pagination and sorting options.

6.3.1.1 Response

The request method provides a status code 200 OK response with a paginated collection of `DeliveryReport` (refer to Table 6-17 Paginated Response).

6.3.2 [GET] Date's Scheduled Messages

This request method provides a list of scheduled messages for a provided date. The HTTP GET "`api/v1/schedules/{year}/{month}/{day}`" route has three parameters to define the specific

`{year}`, `{month}`, and `{day}` of the schedule. The response is also a paginated collection (refer to Figure 6-4 Batch Message Lookup Pagination for additional options) of `DeliveryReport`.

6.3.2.1 Response

The request method provides a status code 200 OK response with a paginated collection of `DeliveryReport` (refer to Table 6-17 Paginated Response).

6.3.3 [PUT] Change Message Scheduled Delivery Date for Batches

The request method HTTP PUT "`api/v1/schedules/{batchId}/batches`" provides a way to update all messages in a batch, specified with the `{batchId}` placeholder, with a new ScheduledDeliveryDate.

6.3.3.1 Request Body

The request method expects a `MessageSchedule` object. Refer to Table 6-15 `MessageSchedule` Object.

6.3.3.2 Response

On success, the response will be a status code 202 Accepted, containing a `Location` header that provides the URI for the updated batch object. If invalid data is provided, status code 400 Bad Request is returned. If an invalid BatchId was provided, then a status code 404 Not Found is returned.

6.4 Subscriber API

Organisations are able to store subscribers through the Administration Web Portal or through a client application integrating the Subscriber API. Organisations have quick access to the registered subscribers or subscribed users to services using the Administration Web Portal. The Subscriber API interface provides CRUD functionality to manage subscriber lists and subscriptions. Table 6-21 lists the allowed request methods for the Subscriber API.

Table 6-21 Subscriber API REST API Endpoints

API	Description
`GET api/v1/subscribers`	Returns a collection of type `Subscriber`.
`GET api/v1/subscribers /{subscriberId}`	Returns a `Subscriber` object for the specified `{subscriberId}`.
`GET api/v1/subscribers /{ subscriberId}/groups`	Returns a collection of the groups subscribed for the user as specified in the `{subscriberId}` placeholder.
`POST api/v1/subscribers`	Creates a new `Subscriber` object.
`PUT api/v1/subscribers /{subscriberId}`	Updates a `Subscriber` object with the `SubscriberId` as specified in the `{subscriberId}` placeholder.
`DELETE api/v1/subscribers /{subscriberId}`	Deletes a `Subscriber` object.

6.4.1 [GET] Get Subscribers

Request method HTTP GET "`api/v1/subscribers`" provides a paginated collection of `Subscriber` objects in the system linked with the client application's API Key Organisation. Refer to section 5 for the additional pagination and sorting parameters.

6.4.1.1 Response

A status code 200 OK is given with both a `Page` (refer to Table 6-18 `Page` Object) object, and a collection of `Subscriber`, detailed below in Table 6-22 Subscriber Object.

Table 6-22 Subscriber Object

Name	Description	Type	Required
`SubscriberId`	Subscriber unique identifier	`integer`	Optional
`Address`	Full Address Details	`string`	Optional
`Locality`	Locality	`string`	Optional
`PostCode`	Post Code	`string`	Optional
`PhoneNo`	Phone Number	`string`	Optional

Name	Description	Type	Required
PhoneNoAlt	Alternate Phone Number	string	Optional
Country	ISO 3166-1 Alpha-2 Country Code	string	Optional
PreferredLanguage	Preferred Language	Language Enum	Required
DisplayName	Display Name	string	Optional
Title	Title	Salutation Enum	Optional
FirstName	First Name	string	Required
LastName	Last Email	string	Required
Email	Email	string	Required
MobileNo	Mobile Number	string	Conditional

6.4.2 [GET] Get Subscriber

HTTP GET "api/v1/subscribers/{subscriberId}" provides the details of a specific Subscriber, based on the {subscriberId} provided.

6.4.2.1 Response

Status code 200 OK is provided if the Subscriber was found, alongside its details (refer to Table 6-22 Subscriber Object). An invalid subscriberId results in a 404 Not Found.

6.4.3 [GET] Get Group Subsriptions

The HTTP GET "api/v1/subscribers/{subscriberId}/groups" route provides a list of groups the particular subscriber has subscriptions to.

6.4.3.1 Response

The response of this method is status code 200 OK, with a collection of Group (defined later on in Table 6-25 Group Object). Incorrect subscriberIds will provide a 404 Not Found.

6.4.4 [POST] Create Subscriber

The HTTP GET "api/v1/subscribers/{subscriberId}/groups" route provides a list of groups the particular subscriber has subscriptions to.

6.4.4.1 Request Body

To create a new `Subscriber`, the client application must provide a `BaseSubscriber`, detailed below in Table 6-23 `BaseSubscriber` Object.

Table 6-23 `BaseSubscriber` Object

Name	Description	Type	Required
`Address`	Full Address Details	`string`	Optional
`Locality`	Locality	`string`	Optional
`PostCode`	Post Code	`string`	Optional
`PhoneNo`	Phone Number	`string`	Optional
`PhoneNoAlt`	Alternate Phone Number	`string`	Optional
`Country`	ISO 3166-1 Alpha-2 Country Code	`string`	Optional
`PreferredLanguage`	Preferred Language	`Language` Enum	Required
`DisplayName`	Display Name	`string`	Optional
`Title`	Title	`Salutation` Enum	Optional
`FirstName`	First Name	`string`	Required
`LastName`	Last Email	`string`	Required
`Email`	Email	`string`	Required
`MobileNo`	Mobile Number	`string`	Conditional

6.4.4.2 Response

If the new `Subscriber` object is created successfully, a status code 201 Created is provided, with a `Location` header URI that can be used to retrieve the new object. Invalid data causes a 400 Bad Request, with a list of ModelState errors.

6.4.5 [PUT] Update Subscriber

Client applications can use the request method HTTP PUT "`api/v1/subscribers/{subscriberId}`", with `{subscriberId}` being a placeholder for the desired reference, to update a `Subscriber`.

6.4.5.1 Request Body

This method accepts a `BaseSubscriber` object (refer to Table 6-23 `BaseSubscriber` Object).

6.4.5.2 Response

If the `Subscriber` is successfully updated, a status code 201 Created is provided, along with the new object's URI in the `Location` header for retrieval. Invalid data causes a 400 Bad Request, and returns a list of ModelState errors, while an invalid subscriberId results in a 404 Not Found.

6.4.6 [DELETE] Delete Subscriber

Client applications can use the request method HTTP PUT "`api/v1/subscribers/{subscriberId}`", with `{subscriberId}` being a placeholder for the desired reference, to update a `Subscriber`.

6.4.6.1 Response

If the `Subscriber` is successfully deleted, a status code 204 No Content is returned. If the `subscriberId` was invalid, or the object has already been deleted, 410 Gone is provided instead.

6.5 Group API

The Notifications Platform provides functionality for organisations to easily manage their subscribers. The system allows for an organisation to create a number of groups that the organisation will support and allocate subscribers to. Individuals can register to a particular subscription group of their preference to receive notifications. From an administrative perspective, this allows an organisation to send a message to an entire subscriber group without having the need to manually input the contact list every time they need to send a group message. Additionally, the system will allow subscribers to opt-out from any of the subscribed service.

The Group API interface provides the necessary functionality to manage the groups list for a particular organisation. It provides basic CRUD functionality and other functions to easily register a user to a particular group or send a message to a group of subscribers. Table 6-24 lists the allowed request methods for the Subscriber API.

Table 6-24 Group API REST API Endpoints

API	Description
`GET api/v1/groups`	Returns a collection of type Group.
`GET api/v1/groups/{groupId}`	Returns a `Group` object `{GroupId, Name}` for the specified `{groupId}`.
`POST api/v1/groups`	Creates a new `Group` object.
`PUT api/v1/groups /{groupId}`	Updates a `Group` object for the specified `{groupId}`.
`DELETE api/v1/groups /{groupId}`	Deletes a `Group` object for the specified `{groupId}`.
`POST api/v1/groups /{groupId}/messages`	Sends a `Message` object to all registered users subscribed to the group as specified in the `{groupId}` placeholder.
`GET api/v1/groups /{groupId}/subscribers`	Get a list of subscribers for the specified `{groupId}`.
`POST api/v1/groups /{groupId}/subscribers`	Create a new `Subscriber` object and register the user to the group as specified in the `{groupId}` placeholder.
`POST api/v1/groups/{groupId} /subscriptions`	Subscribe user to the group as specified in the `{groupId}` placeholder.
`DELETE api/v1/groups/{groupId} /subscribers/{subscriberId}`	Unsubscribe user with `{subscriberId}` from the specified group `{groupId}`.

6.5.1 [GET] Get Organisation Groups

The HTTP GET "`api/v1/groups`" request method provides a list of groups inside an organisation defined by the client application's APIKey.

6.5.1.1 Response

This request method returns a collection of `Group`, as shown in Table 6-25 `Group` Object.

Table 6-25 `Group` Object

Name	Description	Type
`GroupId`	Unique identifier for the `Group` object.	`globally unique identifier`
`Name`	Name of the group.	`string`

6.5.2 [GET] Get Group

This HTTP GET "`api/v1/groups/{groupId}`" request method provides a way to retrieve the details of a specific `Group`, denoted by the `{groupId}` placeholder, the client application's API Key has access to.

6.5.2.1 Response

A `Group` object is returned from this method. Refer to Table 6-25 `Group` Object.

6.5.3 [POST] Create Group

Groups can be created through the HTTP POST "`api/v1/groups`" request method. New groups will be linked to the `Organisation` in the client application's API Key.

6.5.3.1 Request Body

A new Group requires a `BaseGroup` object, as shown in Table 6-26 `BaseGroup` Object below.

Table 6-26 `BaseGroup` Object

Name	Description	Type	Required
`Name`	Group's name.	`string`	Required

6.5.3.2 Response

If the group was successfully created, a status code 201 Created is sent alongside a `Location` header for the URI for the group. Requests with invalid data receive a status code 400 Bad Request instead.

6.5.4 [PUT] Update Group

The "`api/v1/groups/{groupId}`" request method allows the updating of a `Group` object's details after creation, with the id defined in the {`groupId}` placeholder.

6.5.4.1 Request Body

A new Group requires a `BaseGroup` object, as shown in Table 6-26 `BaseGroup` Object.

6.5.4.2 Response

Status code 201 Created is sent alongside a `Location` header for the URI for the group if it was successfully updated. Requests with invalid data receive a status code 400 Bad Request instead.

6.5.5 [DELETE] Delete Group

Group deletion is done via the HTTP DELETE "`api/v1/groups/{groupId}`" request method, with `{groupId}` being the placeholder for the group's Id.

6.5.5.1 Response

The response is a status code 204 No Content if the deletion was successful. If the group was not found or already deleted, a status code 410 Gone is given instead.

6.5.6 [POST] Send Message to Group

HTTP POST "`api/v1/groups/{groupId}/messages`", with `{groupId}` being a placeholder for the group's Id, allows the scheduling of a given message to all subscribers within a group.

6.5.6.1 Request Body

This route expects a `MessageBatchRequest` object, detailed in Table 6-19 `MessageBatchRequest` Object.

6.5.6.2 Response

On successful message creation, a status code 201 Created will contain a MessageBatch object (refer to Table 6-13 `MessageBatch` Object) as well as a `Location` header, which provides the URI for the updated message object. If invalid data is provided (status code 400 Bad Request), the response body will contain the Model State array, listing the errors for each field. If the daily quota in the `Sender` account has been exceeded, a status code 402 Payment Required is sent.

6.5.7 [GET] Get Subscribers in Group

The HTTP GET request method "`api/v1/groups/{groupId}/subscribers`", with `{groupId}` being the group's Id, gives a paginated (refer to section 5.2 **Error! Reference source not found.**for a dditional pagination options) collection of associated `Subscriber` accounts.

6.5.7.1 Response

A status code 200 OK with a collection of `Subscriber` (refer to Table 6-22 Subscriber Object), alongside the appropriate `Page` object (
Table 6-18 `Page` Object), is returned if groupId is valid. Otherwise, a 404 Not Found is provided.

6.5.8 [POST] Create Subscription with a new Subscriber

The HTTP POST "`api/v1/groups/{groupId}/subscribers`" request method allows the creation of a subscription between a `Group`, specified in `{groupId}`, and a new `Subscriber`.

6.5.8.1 Request Body

This route expects a `BaseSubscriber` object, detailed in Table 6-23 `BaseSubscriber` Object.

6.5.8.2 Response

The method returns a status code 201 Created on success, with a `Location` header containing the URI of the new `Subscriber`. Invalid data returns a 400 Bad Request, with a list of ModelState errors. If groupId does not reference any records, 404 Not Found is provided instead.

6.5.9 [POST] Create Subscription with an existing Subscriber

The HTTP POST "`api/v1/groups/{groupId}/subscriptions`" request method allows the creation of a subscription between a `Group`, specified in `{groupId}`, and an existing `Subscriber`.

6.5.9.1 Request Body

This route expects a `Subscription` object, detailed below in Table 6-27 `Subscription` Object.

Table 6-27 `Subscription` Object

Name	Description	Type	Required
`SubscriberId`	Unique identifier of the `Subscriber`.	`globally unique identifier`	Required

6.5.9.2 Response

The method returns a status code 201 Created on success, with a `Location` header containing the URI of the new `Subscriber`. If groupId or subscriberId do not reference any records, 404 Not Found is provided instead.

6.5.10 [DELETE] Delete Subscription

Deleting a subscription is achieved through the HTTP DELETE "`api/v1/groups/{groupId}/subscribers/{subscriberId}`" request method, with `{groupId}` referencing the `Group`, and `{subscriberId}` referencing the `Subscriber`.

6.5.10.1 Response

If both groupId and subcriberId reference existing records and deletion is successful, status code 204 No Content is returned. Otherwise, 410 Gone is provided.

6.6 Attachment API

The Attachment API allows for the direct access to attachments created as parts of messages in the Notifications Platform.

Table 6-28 Attachment API REST API Endpoints

API	Description
`GET api/v1/attachments/{attachmentId}`	Returns a file that was attached to a message.

6.6.1 [GET] Get Attachment File

The HTTP GET "`api/v1/attachments/{attachmentId}`" request method is used to obtain the file referenced by an `{attachmentId}`. Attachments are initialized during message creation operations.

6.6.1.1 Response

A correct attachmentId will return a status code 200 OK with the raw stream of bytes of the file. If the `Attachment` was not found, 404 Not Found is returned.

6.7 Key API

The Key API handles the management of APIKeys used by the client applications. Client applications must interact with this API to refresh their APIKeys regularly.

Table 6-29 API Key API REST API Endpoint

API	Description
`GET api/v1/key`	Invalidates the current Secret Key and generates a new key.

6.7.1 [GET] Refresh APIKey

The client application should anticipate the possibility that a granted key might no longer work and the application is tasked in initiating a request for a new Secret Key. To resolve this, a request for a new key using the HTTP GET "`api/v1/key`" request method should be made after the client application uses the existing API Key and Secret Key pair to generate the `Authorization` header, using the "SMG-V1-HMAC-SHA256" scheme. The server looks up the API Key from the operation database, invalidates the existing Secret Key, generates a new key, and returns it.

6.7.1.1 Response

This route returns 200 OK, with a new `APIKey` object, if the refresh is successful (as detailed in below).

Table 6-30 `APIKey` Object

Name	Description	Type
`Name`	Friendly name to identify APIKey.	`string`
`Key`	Unique identifier for the key.	`string`
`Secret`	Secret Key (must be refreshed every seven days or as per the indicated Expiry Date field).	`string`
`ExpiryDate`	Expiry Date. If null the Secret Key never expires (only used for testing).	`date`

6.8 Message Pricing API

The Message Pricing API can be used to calculate an estimate for the total cost to send a new message. Moreover, this can be integrated in a client application to preview the message prior sending e.g. for SMS type messages, the API will return the total number of characters in the message text, a preview of how the message will be split according to the SMS standard (including the number of message parts), the encoding type detected from the message text, price per message and the total cost based of the total number of recipients and the type of message being sent. In the case of SMS messages, the cost of the message will be based on the recipient's mobile number dialling prefix, the length of the message and the type of encoding detected. The returned object will also contain a breakdown of how the total cost for each message is calculated.

Pricing request are sent by the client application using an HTTP POST request method through the "`api/v1/message-pricing`" REST API endpoint as shown in Table 6-31. The endpoint supports both SMS and Email type messages, and the total cost will be calculated based on the pricing information in the system for each message type.

Table 6-31 Message Pricing REST API Endpoint

API	Description
`POST api/v1/message-pricing`	Returns pricing details based on the provided message information and a breakdown of how the total cost is calculated.

To request pricing information for a message, the client the client application is to compile and send a `MessageQuote` object using the Message Pricing endpoint as shown in Table 6-31. Table 6-32 shows the structure of the `MessageQuote` object. This is similar to the `Message` object, discussed in the previous section, missing some fields which are not required to compute the cost of the message.

Table 6-32 `MessageQuote` Object

Name	Description	Type	Required
`SenderId`	Sender Identifier used by the Notifications Platform to specify the sender's name for the message.	`globally unique identifier`	Required
`MessageType`	Message Type E.g. Email or SMS	`MessageType Enum`	Required
`MessagePriority`	Message Priority – Low, Normal, and High. These are used by the Message Broker to prioritize message delivery.	`MessagePriority Enum`	Required

Name	Description	Type	Required
Contacts	The Contacts property is a collection of subscribers to specify the delivery address where the message is sent. Depending on the message type, this can be either an MSISDN or Email address.	Collection of Contact	Required
MessageContent	A collection of MessageContent that defines the content of the messages to be delivered.	Collection of MessageContent	Required

Upon submission of the MessageQuote object, using the HTTP POST request method, if the request was successful the API will return a status code 200 OK with a collection of type MessageReceipt as shown in Table 6-33. Since the message to be sent may fall under different pricing schemes, the collection of type MessageReceipt will include pricing information based on the type of message, the recipient's destination address, language or type of encoding used to compile the message, and the length or size of the message.

If the provided information in the MessageQuote object is invalid (status code 400 Bad Request), the response body will contain the list of validation errors for each respective field.

Table 6-33 MessageReceipt Object

Name	Description	Type
Country	ISO 3166-1 Alpha-2 Country Code where the message will be sent. The Country code will only be used for the SMS Type Messages. This identifies a group of messages having the recipients' dialling prefix set to this country.	string
Language	Message Language. Note for SMS Type Messages, languages that use UTF encoding (e.g. Maltese) uses 16 bits per character which will limit 1 SMS to a maximum of 70 characters and concatenated messages to 67 characters.	Language
CharacterCount	The number of characters in the message. For SMS type messages, depending on the "Encoding" of the text provided, SMS text messages are limited to either 160 (GSM-7 encoding equating to 7 bits per character) or 70 (UTF encoded messages, such as Maltese, using 16 bits per character) characters in length. If a message length exceeds 160 characters in case of 7-bit encoding (or 70 characters for UTF encoding), the message is split up to multiple separate SMS and sent to the handset separately, to be concatenated on the receiver's end. For GSM-7 encoded long messages, exceeding 160 characters, these are split into 153 character chunks (7 characters used for segmentation information and for concatenation individual messages back together). For UTF encoded long messages, they are split into 67 character chunks (with 3 characters used for segmentation information and to concatenate the individual messages back together). Note that the standard SMS message can only contain a maximum of 1120 bits, therefore messages will be limited to 7 concatenated messages.	integer

Name	Description	Type
MessagePartsCount	The number of messages the message text will be split into. For SMS Type Messages, MessageCount will range from 1 to 7, while for Email Type Messages, the MessageCount property will always be set to 1.	byte
MessagePartMaxCharacters	The calculated maximum number of characters for each Message Part, based on the message text encoding. This field will only be populated for SMS Type Messages.	integer
MessageParts	This provides a preview for SMS Type Messages of how the message text will be split into parts as per the SMS standard.	Collection of string
Encoding	The type of encoding detected from the provided Message Content as follows: • ASCII: An encoding for the ASCII (7-bit) character set. For SMS Type Messages, Latin-based languages such as English, French, and Spanish use GSM encoding, which equates to 7 bits per character; limiting 1 SMS to at most 160 characters. • Unicode: An encoding for Unicode Transformation Format (UTF-8 or UTF-16) format using the little-endian byte order. For SMS Type Messages, languages that use UTF encoding such as Maltese uses 16 bits per character; limiting 1 SMS to a maximum of 70 characters.	Encoding
MessagePrice	The Price per Message in Euro.	decimal number
TotalRecipientsCount	The total number of recipients that the message will be sent. Recipients will be grouped by the Country Dialling Code provided.	integer
TotalMessagesCount	The actual total number of messages to be sent. For SMS Type Messages, this is the total MessagePartsCount multiplied by the TotalRecipientsCount.	integer
TotalCost	The Total Cost to send the Message in Euro.	decimal number
WarningMessages	Warning Messages provided by the Notifications System to be displayed by the client application.	Collection of string

7 Appendix

7.1 Web API Samples

7.1.1 Example JSON Request

The following is an example of the JSON request structure expected by the Notifications Platform REST Web API, adhering to industry standards.

```
{
  "Contacts": [
    {
      "DisplayName": "John Doe",
      "Title": "Mr",
      "FirstName": "John",
      "LastName": "Doe",
      "Email": "johndoe@gov.mt",
      "MobileNo": ""
    }
  ],
  "MessageContent": [
    {
      "Language": "en",
      "Subject": "Test Subject",
      "MessageBody": "Test Body",
      "Attachments": [
        {
          "ContentStream": "QEA=",
          "FileName": "testfile.txt",
          "ContentType": "text/plain"
        }
      ]
    }
  ],
  "ClientReference": "3aad2777-3091-4f32-9f86-ab297505f0b0",
  "MessageType": "email",
  "MessagePriority": "100",
  "SenderId": "dd024a9b-ca59-4ad9-a9ee-e99e7deba52d",
  "CallbackUrl": "127.0.0.1:8080/message/response",
  "ScheduledDeliveryDate": "2016-04-28T14:14:54.4117761+02:00"
}
```