Scurri

Tracking - Integrator's Guide

2019-03-19

Table of Contents

Introduction

The goal of this document is to provide enough information for Scurri customers to consume tracking data.

Scurri's tracking solution maps carriers' tracking statuses to a specific set of statuses, which are:

- `UNKNOWN` - Carrier's status is not mapped to any of the Scurri statuses
- `MANIFESTED` - Shipment has been manifested
- `IN_TRANSIT` - Shipment is in transit
- `OUT_FOR_DELIVERY` - Shipment out for delivery
- `DELIVERED` - Shipment delivered
- `EXCEPTION` - Some exception happened during the delivery process

These statuses are referred to as unified Scurri status throughout this document.

Next to the unified status you'll also find the carrier's original status code, location and description in the events, which are not mapped by Scurri and thus vary by carrier.

Prerequisites

Scurri's tracking system can only track parcels created using Scurri's main shipping solution, therefore to use tracking you have to have a live production account on `data.scurri.co.uk` and your company needs to have the tracking feature enabled. To achieve this, please contact us at `support@scurri.com` or via your Account Manager/Sales Representative.

Integration Methods

If you want to get tracking events, you have several options: you can poll tracking events through the API, you can specify a webhook that the tracking system will call whenever new tracking events are availabe, or you can subscribe an SQS queue to your designated SNS tracking topic.

Data Format

Both the API and Webhook is using JSON data format. Please always include `Content-Type: application/json` header in your requests.

API Integration

Prerequisites

To access the tracking API, you would need to get an account: contact us with the following information:

- Your company's name in the Scurri system
- A valid e-mail address (which will be used as a login)

Endpoints

Once you have your login details to the tracking system, you can start making API calls.

The tracking system's API is available at:

- `https://tracking.scurri.co.uk/api/v1`

Staging server's address is:

- `https://tracking-staging.scurri.co.uk/api/v1`

This document will refer to this URL as BASE URL for the API. If you execute the shell commands in this document, they all presume that `BASE_URL` environment variable is available:

```
export BASE_URL="https://tracking.scurri.co.uk/api/v1"
```

URLS

The v1 API supports the following urls:

- POST `/authorizations`
- GET `/carriers`
- GET `/carriers/<carrier-slug>`
- GET `/carriers/<carrier-slug>/trackings`
- GET `/carriers/<carrier-slug>/trackings/<tracking-no>`
- GET `/trackings`
- GET `/trackings/<package-id>`
- POST `/webhook-testing`

Authentication

For making API calls, you first need to POST your username/password to `/authorizations` in a JSON structure to retrieve your authentication token:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -d '{"username":"user","password":"password"}' \
    "${BASE_URL}/authorizations"
```

Which will result in a response like:

```
{"token":"f264008eb2c30cacae372c72bf002c3f1d4bad61"}
```

For further API requests, you'll need to send this token as HTTP Authorization header in the following format:

```
Authorization: Token <your-token>
```

The `curl` examples in this document will assume that the token is exported as an environment variable:

```
export TOKEN="f264008eb2c30cacae372c72bf002c3f1d4bad61"
```

Pagination

When you query data through the API, the number of returned items might be large. The API is using pagination, so the maximum number of items returned by the API is limited to `100`. The response will contain a link to retrieve the next/previous page of the results. For first/last pages of data the previous/next links will be null. In general each listing response will have the following structure to support pagination:

```
{
  "count": 1000,
  "next": "https://.../api/v1/carriers/colissimo/trackings?page=2",
  "previous": null,
  "results": [
      ...
  ]
}
```

Where:

- `count`: the total number of results matching your query
- `next`: link to retrieve the next set of results
- `previous`: link to retrieve the previous set of results
- `results`: array containing results

Error Codes

If a specific resource cannot be found, you'll get a `404` status code and the following JSON payload:

```
{
  "detail": "Not found."
}
```

Data Structures

This section contains description of the different data structures you might get back in a response.

Carrier

A carrier object has the following attributes:

- `slug` : use this string to refer to the carrier, 30 chars
- `name` : name of the carrier, 50 chars
- `url` : a URL where you can get the details of the carrier
- `trackings_url` : use this url for tracking events belonging to this carrier

Package

A package object has the following attributes:

- `id` : internal id for the specific parcel, integer
- `tracking_number` : tracking number for the actual parcel, 50 chars
- `url` : a URL where you can get the details of the package
- `created_at` : when the package was created/imported to the tracking system, formatted as `YYYY-MM-DDTHH:MM:SS+HH:MM`
- `carrier` : name of the carrier, see carrier for data length
- `carrier_url` : an URL pointing to the related carrier resource
- `events` : list of tracking events related to this package (see below)

Event

- `id` : internal id for the event, integer
- `status` : unified Scurri status for this event, see unified statuses for possible values.
- `carrier_code` : status code provided by the carrier, 50 chars
- `description` : carrier provided description for this specific event, 300 chars.
- `timestamp` : date/time when the event happened. This time has no localization (timezone), thus it should be interpreted in the timezone appropriate for the carrier in question. Formatted as `YYYY-MM-DDTHH:MM:SS`
- `location` : location related to the event (if provided by carrier), 40 chars

List Carriers

To list available carriers, get the `/carriers` resource:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/carriers
```

The response will contain the list of carrier objects in the `results` field of the response:

```
{
  "results": [
    {
      "slug": "colissimo",
      "name": "Colissimo",
      "url": "https://.../api/v1/carriers/colissimo",
      "trackings_url": "https://.../api/v1/carriers/colissimo/trackings"
    },
    ...
```

Carrier Details

To get the details of a specific carrier, GET the resource specified by the `url` field on the carrier list, or use the known carrier slug to construct the url `/carriers/<carrier-slug>`:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/carriers/colissimo
```

returns a single carrier object:

```
{
  "slug": "colissimo",
  "name": "Colissimo",
  "url": "https://.../api/v1/carriers/colissimo",
  "trackings_url": "https://.../api/v1/carriers/colissimo/trackings"
}
```

All Tracking Events for a Specific Carrier

To get tracking events for a specific carrier, use the url provided by the `trackings_url` field of the carrier detail/listing, or use the carrier slug to construct a `/carriers/<carrier-slug>/trackings` url:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/carriers/colissimo/trackings
```

This returns trackings belonging to the carrier and the relevant events embedded:

```
{
  "results": [
    {
      "id": 39,
      "tracking_number": "0000000000",
      "url": "https://.../api/v1/carriers/colissimo/trackings/0000000000",
      "created_at": "2019-02-11T15:19:06+00:00",
      "carrier": "Colissimo",
      "carrier_url": "https://.../api/v1/carriers/colissimo",
      "events": [
        {
          "id": 2,
          "status": "MANIFESTED",
          "carrier_code": "Some-carrier-code",
          "description": "Some-description",
          "timestamp": "2019-02-11T15:19:06",
          "location": "Location"
        }
      ]
    },
    ...
  ]
}
```

All Tracking Events

To retrieve all tracking events for your subscription, GET `/trackings`:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/trackings
```

Which returns all the tracking objects with events embedded:

```
{
  "results": [
    {
      "id": 2257,
      "tracking_number": "0000000000",
      "url": "https://.../api/v1/trackings/2257",
      "created_at": "2019-02-17T21:26:51+00:00",
      "carrier": "Colissimo",
      "carrier_url": "https://.../api/v1/carriers/colissimo",
      "events": [
        {
          "id": 2110,
          "status": "MANIFESTED",
          ...
```

Events for One Parcel by ID

To retrieve information for a specific parcel, you would need to use its `id` to construct `/trackings/<parcel-id>`:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/trackings/2257
```

Which returns a package object:

```
{
  "id": 2257,
  "tracking_number": "0000000000",
  "url": "https://.../api/v1/trackings/2257",
  "created_at": "2019-02-17T21:26:51+00:00",
  "carrier": "Colissimo",
  "carrier_url": "https://.../api/v1/carriers/colissimo",
  "events": [...]
}
```

Events for One Parcel by Tracking Number

If you know the tracking number and the carrier's slug, use

`/carriers/<carrier-slug>/trackings/<tracking-number>`:

```
curl -X GET \
    -H "Authorization: Token ${TOKEN}" \
    ${BASE_URL}/carriers/colissimo/trackings/0000000000
```

Which returns a package object:

```
{
  "id": 2257,
  "tracking_number": "0000000000",
  "url": "https://.../api/v1/carriers/colissimo/trackings/0000000000",
  "created_at": "2019-02-17T21:26:51+00:00",
  "carrier": "Colissimo",
  "carrier_url": "https://.../api/v1/carriers/colissimo",
  "events": [
    {
      "id": 2110,
      "status": "MANIFESTED",
      "carrier_code": "Some-carrier-code",
      "description": "Some-description",
      "timestamp": "2019-01-02T12:32:43",
      "location": "Location"
    }
  ]
}
```

Webhook Integration

If you want to be notified of new tracking events, you can use the webhook method. This means that you'll need to implement a https endpoint yourself and let Scurri configure its details in the tracking system, so tracking events will be posted to the endpoint.

The tracking system will send you a POST request with the following JSON structure:

```
{
   "carrier_slug":"collectplus",
   "count":9,
   "trackings":[
      {
         "events":[
            {
               "timestamp":"2019-01-24T15:18:47",
               "carrier_code":"ACC",
               "description":"Received at BARRY'S ...",
               "location":"",
               "status":"IN_TRANSIT"
            }
         ],
         "tracking_number":"826FG54228",
         "id":1079,
         "carrier":"Collect+"
      },
      ...
```

Where:

- `carrier_slug` : slug for the carrier to whom the updates belong - str(30)
- `count` : Number of parcels within the payload - int
- `trackings` : List of:
- `tracking_number` : tracking number of the package - str(50) **required**
- `id` : package ID (can be used via the API, see Events for One Parcel by ID) - int **required**
- `carrier` : name of the carrier - str(50) **required**
- `events` : List of new events for the package:
  - `timestamp` : timestamp for the event - "%Y-%m-%dT%H:%M:%S" format **required**
  - `carrier_code` : code for the event provided by the carrier - str(50)
  - `description` : carrier provided description of the event - str(300)
  - `location` : location of the event - str(40)
  - `status` : unified Scurri status for this event **required**

Each payload will contain data for up to 100 parcels.

Supported Authentications

For the tracking system to authenticate, you can use one of the following:

Basic Authentication

Standard basic authentication using a username and a password. For this you'll need to specify:

- URL of your webhook
- a username
- a password

Token Based Basic Authentication

The `Authorization` http header that the tracking system sends consists of:

- the string `Basic`
- followed by a single space: ` `
- followed by the base64 encoded string of: token + `:` (colon character)

In pseudo-code:

```
auth_header = "Basic " + b64encode(token + ":")
```

To retrieve the authentication token for a specific request that the tracking system sends, you would want to:

- get the value of the `Authorization` HTTP header
- cut off `"Basic "` from the beginning
- base64 decode the rest
- cut off `":"` from the end of the string
- you are left with the authentication token, which you can compare with your secrets.

If your service is using this authentication method, you'll need to specify the following:

- URL of your webhook
- a token

Retry Mechanisms

If the resulting http error code is different from `404`, the system will re-try to post the same payload after one second, then give up.

Webhook Testing

If you would like to manually test the your integration, you can do so by making a POST request to the webhook testing endpoint found at `/webhook-testing`

To do this, you must post a valid JSON payload of data:

```
curl -i -X POST \
    -H "Authorization: Token ${TOKEN}" \
    -H "Content-Type: application/json" \
    ${BASE_URL}/webhook-testing \
    -d @- << EOF
{
    "carrier_slug": "collectplus",
    "count": 1,
    "trackings": [
        {
            "events": [
                {
                    "timestamp": "2019-01-24T15:18:47",
                    "carrier_code": "ACC",
                    "description": "Received at ...",
                    "location": "some-location",
                    "status": "IN_TRANSIT"
                }
            ],
            "tracking_number": "826FG54228",
            "id": 1079,
            "carrier": "Collect+"
        }
    ]
}
EOF
```

This payload needs to contain the same data structure as what is posted to the webhook via the webhook integration method. See the Webhook Integration section of this documentation to for more information.

Note: The maximum amount of `trackings` that are allowed is 100, and each tracking must contain no more than 10 `events`. You can find a list of the Scurri specific statuses at the beginning of this document.

When the call to the api endpoint is made, the system will validate the data. If the validation is successful it will make a call to your configured webhook(s) and will post the data you have provided.

If the call is successful you will receive a HTTP status code of 200 and the following JSON response:

```
{
    "test_result":
        {
            "errors": [],
            "webhook_calls": ["<webhook_url>"],
            "success": true
        }
}
```

If the call is unsuccessful you will receive a HTTP status code of 400 and the following JSON response:

```
{
    "test_result":
        {
            "errors": [
                "There was an issue making a call to <webhook_url> (<error>)"
            ],
            "webhook_calls": ["<webhook_url>"],
            "success": false
        }
}
```

SNS Integration

As an alternative to webhook integration we also offer an AWS based SNS integration where Scurri creates an SNS topic for your tracking updates to which you can subscribe your SQS queue.

To get started with this approach please send us the following details:

- Your AWS account id

Once Scurri has created the SNS topic for your tracking updates, you'll receive the ARN of the topic to which you can subscribe your SQS queues. See Example: Subscribing to Tracking Updates SNS Topic section of the Appendix for a CloudFormation template.

Appendix

Example: Subscribing to Tracking Updates SNS Topic

Here is a CloudFormation template which can be used to create an SQS queue and subscribe that to your tracking topic:

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "SQS queue to listen to tracking updates",
  "Parameters": {
    "Topic": {
      "Description": "ARN of remote queue",
      "Type": "String"
    }
  },
  "Resources": {
    "Queue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
      }
    },
    "QueueAccessPolicy": {
      "Type": "AWS::SQS::QueuePolicy",
      "Properties": {
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "AWS": "*"
              },
              "Action": "SQS:SendMessage",
              "Resource": {
                "Fn::GetAtt": [
                  "Queue",
                  "Arn"
                ]
              },
              "Condition": {
                "ArnEquals": {
                  "aws:SourceArn": {
                    "Ref": "Topic"
                  }
                }
              }
            }
          ]
        },
        "Queues": [
          {
            "Ref": "Queue"
          }
        ]
      }
    },
```

(Continued on the next page)

```
    "Subscription": {
      "Type": "AWS::SNS::Subscription",
      "Properties": {
        "Endpoint": {
          "Fn::GetAtt": [
            "Queue",
            "Arn"
          ]
        },
        "Region": "eu-west-1",
        "Protocol": "sqs",
        "RawMessageDelivery": true,
        "TopicArn": {
          "Ref": "Topic"
        }
      }
    }
  },
  "Outputs": {
    "QueueURL": {
      "Description": "URL of newly created SQS Queue",
      "Value": {
        "Ref": "Queue"
      }
    },
    "QueueARN": {
      "Description": "ARN of newly created SQS Queue",
      "Value": {
        "Fn::GetAtt": [
          "Queue",
          "Arn"
        ]
      }
    },
    "QueueName": {
      "Description": "Name of newly created SQS Queue",
      "Value": {
        "Fn::GetAtt": [
          "Queue",
          "QueueName"
        ]
      }
    }
  }
}
```

If you saved this file as `scurri-subscription.json`, you can create a stack using `awccli`:

```
aws cloudformation \
    create-stack \
        --capabilities CAPABILITY_IAM \
        --stack-name "scurri-tracking-updates" \
        --template-body "file://scurri-subscription.json" \
        --parameters ParameterKey=Topic,ParameterValue=<SNS-TOPIC>
```

where `<SNS-TOPIC>` is the topic ARN given to you by Scurri.