ONE Record API & Security

Reference Specification – draft

March 9, 2020

Note on this draft

This reference specification is a draft. This means that it is subject to edits. Discussion on this specification is highly encouraged and please contact onerecord@iata.org with any comments or suggested improvements.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change Log** | | | | | | | |
| **Change No.** | **Description of Change** | **Requestor** | **Date Submitted** | **Date Approved** | **Status** | **Comments** |
| 1 | Add scenario 2 for Publisher/Subscriber (process initiated by the subscriber) | A. Blaj | 2019/10/16 |  | Pending |  |
| 2 | Add delegation model | A. Blaj | 2019/10/16 |  | Pending |  |
| 3 | Rename serverInformation to companyInformation. Add companyIdentifier | A. Blaj | 2019/11/07 |  | Pending |  |
| 4 | Add notification request model (pub/sub) | A. Blaj | 2019/12/19 |  | Pending |  |
| 5 | Add logisticsObjectRef field to audit trail model | A. Blaj | 2020/01/22 |  | Pending |  |
| 6 | Events (statuses) specifications | A. Blaj | 2020/03/09 |  | Pending |  |

Contents

[Overview 4](#_Toc34661856)

[ONE Record Server API 5](#_Toc34661857)

[Logistics Object ID 5](#_Toc34661858)

[Create Logistics Object 5](#_Toc34661859)

[Read Logistics Object 6](#_Toc34661860)

[Update Logistics Object 7](#_Toc34661861)

[Error model 11](#_Toc34661862)

[ONE Record Security 14](#_Toc34661863)

[Identity and Authentication Providers (IAP) 14](#_Toc34661864)

[Authentication 15](#_Toc34661865)

[Token Verification 18](#_Toc34661866)

[Authorization 18](#_Toc34661867)

[Publish & Subscribe with ONE Record 20](#_Toc34661868)

[Publish & Subscribe model 20](#_Toc34661869)

[The ONE Record Company Identifier 23](#_Toc34661870)

[Get Company Information from the IoL 24](#_Toc34661871)

[Get Subscription Information 26](#_Toc34661872)

[Subscriptions 27](#_Toc34661873)

[Notification Request 28](#_Toc34661874)

[ONE Record Client Subscription API 28](#_Toc34661875)

[Delegation 30](#_Toc34661876)

[Events (status update) in ONE Record 32](#_Toc34661877)

[Create event 32](#_Toc34661878)

[Retrieve events 33](#_Toc34661879)

[Glossary 35](#_Toc34661880)

# Overview

ONE Record specifies the API and security model for data exchange over the Internet of Logistics. In fact, ONE Record is essentially the *specification* of the *Internet of Logistics*. The **Internet of Logistics** or **IoL** is the collection of all ONE Record Clients and Servers.

In the Internet of Logistics companies can exchange data as needed. They can host and publish Logistics Objects on ONE Record Servers and their partners can access these Logistics Objects using ONE Record Clients. Logistics Objects are also created or updated using this same ONE Record Server API.

ONE Record Client

ONE Record Server

API

In order to optimize the data flows in this Internet of Logistics, ONE Record provides for a Publish & Subscribe model. This requires that the ONE Record Clients implement a subscription API to which the ONE Record Server can publish new and updated Logistics Objects

ONE Record Client

ONE Record Server

API

API

Subscriber Publisher

This document is in three parts. The first part specifies the ONE Record Server API, the second part concerns the security model and the third part covers the Publish-Subscribe model which includes the ONE Record Client API.

# ONE Record Server API

The ONE Record Server API is a REST based API that supports the following operations:

* Create Logistics Objects
* Read Logistics Objects
* Patch Processing Updates to Logistics Objects

ONE Record Client

ONE Record Server

API

## Logistics Object ID

A Logistics Object can be identified by a Logistics Objects ID.  A LO ID can be any URL which is unique by definition. An example of a LO ID could be for example:

https:// {ORS Domain} / {license plate} / {unique id to identify LO}

where

{ORS Domain} - The domain name associated with the ONE Record Server e.g. www.onerecordcargo.org

{license plate} - The company identifier for this ONE Record Server, e.g. my\_airline

{unique ID to identify LO} - An identifier for the Logistics Object that is unique at least for this server.

An example of a LO ID is: https://www.onerecordcargo.org/my\_airline/airwaybill\_123-12345678

The LO ID should be URL friendly, i.e. avoid unsafe characters that include the blank/empty spaces and " < > # % { } | \ ^ ~ [ ] `.

## Create Logistics Object

Publishes a Logistics Object resource to a ONE Record Server.

The user creating a Logistic Object must have authorization to create Logistics Objects and must belong to the company that is identified by the license plate in the LO ID.

### Request

HTTP Request type: POST

### HTTP Headers

The following HTTP header parameters MUST be present in the POST request:

|  |  |
| --- | --- |
| **Authentication** | A valid Bearer Token that is provided by the Identity and Authentication Provider. Refer *ONE Record security* |
| **Accept** | The content type that you want the HTTP response to be formatted in. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |
| **Content-Type** | The content type that is contained with the HTTP body. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |

### HTTP Body

The HTTP body MUST be a valid supported Logistics Object in the format as specified by the Content-Type in the header.

The full specification of the Logistics Objects can be found here: [https://github.com/IATA-Cargo/ONE-Record/](https://github.com/IATA-Cargo/ONE-Record/tree/master/ontology)

### Response

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 201 | Logistics Object has been published to the Internet of Logistics. | No body required |
| 400 | Invalid Logistics Object | Error model |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to publish the Logistics Object to the Internet of Logistics | Error model |
| 415 | Unsupported Content Type | Error model |

## Read Logistics Object

Retrieves a Logistics Object resource from a ONE Record Server.

The user performing the GET request must belong to a server that has been given access to the Logistics Object.

### Request

HTTP Request type: **GET**

The request URL should contain the Logistics Objects ID to be retrieved.

### HTTP Headers

The following HTTP header parameters MUST be present in the GET request:

|  |  |
| --- | --- |
| Authentication | A valid Bearer Token that is provided by the Identity and Authentication Provider. Refer *ONE Record security* |
| Accept | The content type that you want the HTTP response to be formatted in. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |

### Response

A positive HTTP 200 response is expected to a GET request. The body of the response is expected to be the Logistics Object in the format that has been requested in the Accept header of the request.

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 200 | The request to retrieve the Logistics Object has been successful | Logistics Object |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to retrieve the Logistics Object | Error model |
| 404 | Logistics Object not found | Error model |

## Update Logistics Object

The PATCH request should be used to provide updates on the Logistics Object. The user performing the PATCH must belong to a company that is authorized to access to the Logistics Object.

Update Logistics Objects follow the specifications from: <https://www.w3.org/TR/ldpatch/> and <https://github.com/digibib/ls.ext/wiki/JSON-LD-PATCH>.

Some considerations related to updating Logistics Objects:

* Only publisher can change the Logistics Object, where publisher is the party that creates the Logistics Object on the ONE Record server. The publisher can be considered also the owner of the Logistics Object.
* A business partner can request a change on the Logistics Object.
* The publisher will make the Logistics Object changes based on the defined business rules.
* An audit trail of all the changes will be stored and can be retrieved at any moment from a dedicated endpoint on the ONE Record Server (e.g. https://www.onerecordcargo.org/my\_airline/airwaybill\_123-12345678/auditTrail).
* Logistics Objects should have a revision number, which is an integer to be incremented after every applied change.
* When retrieving a Logistics Object, the latest version should be returned always.
* Evaluation of a PATCH document occurs as a single event. Operations are sorted and processed as groups of delete and then add operations until the operations are applied, or the entire PATCH fails. Meaning that if a field update fails, the whole PATCH request is unsuccessful. No partial updates should be accepted.
* As a best practice, a GET Logistics object should be performed before requesting a PATCH in order to make sure that the change is made towards the latest version of the object.
* If a change request is rejected, the revision number of the Logistics Object is not incremented.
* TODO: Should rejected update requests be kept in the audit trail of the Logistics Object?

### Request

HTTP Request type: PATCH

The request URL should contain the Logistics Objects ID to be updated.

### HTTP Headers

The following HTTP header parameters MUST be present in the PATCH request:

|  |  |
| --- | --- |
| Authentication | A valid Bearer Token that is provided by the Identity and Authentication Provider. Refer *ONE Record security* |
| Accept | The content type that you want the HTTP response to be formatted in. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |
| Content-Type | The content type that is contained with the HTTP body. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |

### HTTP Body

The HTTP body MUST be a valid array of objects; each object represents a single operation to be applied to a target linked data resource. The format should be as specified by the Content-Type in the header.  The following is an example of supported PATCH request body:

|  |
| --- |
| {     **"revision"**:"1",    **"description"**:"Reason for the change (optional)",    **"operations"**:[        {           **"op"**:"del",          **"p"**:"[http://cargo.iata.org/AirWaybill#totalPieceAndULDCount](https://tcfplayground.org/totalPieceCount)",          **"o"**:{              **"value"**:"10",             **"datatype"**:"<https://www.w3.org/2001/XMLSchema#decimal>"          }       },        {           **"op"**:"add",          **"p"**:"[http://cargo.iata.org/AirWaybill#totalPieceAndULDCount](https://tcfplayground.org/totalPieceCount)",          **"o"**:{              **"value"**:"11",             **"datatype"**:"<https://www.w3.org/2001/XMLSchema#decimal>"          }       },       {           **"op"**:"add",          **"p"**:" <http://cargo.iata.org/Airwaybill#date>",          **"o"**:{              **"value"**:"2019-08-18",             **"datatype"**:"http://www.w3.org/2001/XMLSchema#date"          }       }    ] } |

* Operation objects must have exactly one "op" (operation) member; this value indicates which operation is to be performed. The value must be one of "add" or "del"; all other values result in an error.
* Operations objects must have exactly one "p", predicate, member. The value of this member must be an IRI.
* Operations objects must have exactly one "o", object, member. The value of this member must an object. This object must contain two members, a "value" and a "datatype".
* TODO: Add provenance information? (<https://www.w3.org/TR/prov-o/>)

Operations:

* **add**

Add has a very simple function, it always adds new sets of statements. If a pre-existing statement exists with similar or the same characteristics, it must not be overwritten. To overwrite, a delete and an add operation must be performed.

* **del**

Del also has a very simple function, it always removes sets of statements.

### Response

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 204 | The update has been successful | No body required |
| 400 | The update is invalid | Error model |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to update the Logistics Object | Error model |
| 404 | Logistics Object not found | Error model |
| 415 | Unsupported Content Type, response when the client sends a PATCH document format that the server does not support for the resource identified by the Request-URI. | Error model |
| 422 | Unprocessable request, when the server understands the patch document and the syntax of the patch document appears to be valid, but the server is incapable of processing the request. | Error model |

### Logistics object audit trail

TODO: Add signature per field?

A Logistics Object audit trail can be retrieved by performing a GET request to:

https:// {ORS Domain} / {license plate} / {unique id to identify LO} /auditTrail

In order to retrieve the history of a Logistics Object after a given date, a query parameter should be added to the request URL as follows:

https:// {ORS Domain} / {license plate} / {unique id to identify LO} /auditTrail?updatedFrom=YYYYMMDDThhmmssZ&updatedTo=YYYYMMDDThhmmssZ

|  |
| --- |
| {     **"create"**:{        **"lo"**:"initial content of the Logistics Object"    },,  **"logisticsObjectRef"**:"Logistics Object Id to which the audit trail applies",     **"changeRequests"**:[        {           **"timestamp"**:"2019-09-17T14:49:13+00:00",          **"companyId"**:"http://myonerecordserver.com/AIRLINE",          **"changeRequest"**:{              **"revision"**:"1",             **"description"**:"Updated number of total pieces count",             **"operations"**:[                 {                    **"op"**:"del",                   **"p"**:"http://cargo.iata.org/AirWaybill#totalPieceAndULDCount",                   **"o"**:{                       **"value"**:"10",                      **"datatype"**:"https://www.w3.org/2001/XMLSchema#decimal"                   }                },                {                    **"op"**:"add",                   **"p"**:"http://cargo.iata.org/AirWaybill#totalPieceAndULDCount ",                   **"o"**:{                       **"value"**:"11",                      **"datatype"**:"https://www.w3.org/2001/XMLSchema#decimal"                   }                },                {                    **"op"**:"add",                   **"p"**:" http://cargo.iata.org/AirWaybill#date",                   **"o"**:{                       **"value"**:"2019-08-18",                      **"datatype"**:"http://www.w3.org/2001/XMLSchema#date"                   }                }             ]          },          **"errorIndicator"**:[              {                 **"error"**:"error model"             }          ],          **"status"**:"ACCEPTED"       }    ] } |

## Error model

This section describes the datatype definitions used within the ONE Record API for error handling.

### Error HTTP Status Codes

The following table contains a non-exhaustive list of HTTP error statuses that require an error model response:

|  |  |
| --- | --- |
| Code | Description |
| 400 | Bad request |
| 401 | Unauthorized or expired token |
| 403 | Forbidden to perform action |
| 404 | Not Found |
| 405 | Method Not Allowed |
| 415 | Unsupported Content Type |
| 500 | Internal Server Error |

### Error response headers

The error response should always contain the **HTTP Status** and the **Content-Type header**:

HTTP/1.1 400 Bad Request

Content-Type: application/ld+json

### Error Payload

The error response should contain the following fields:

|  |
| --- |
| {  "@context": {  "@vocab: "http://cargo.iata.org/"  },  "@type": "Error",  "title": "Request contains invalid field",  "details": [{  "code": "1234",  "attribute": ".AirWayBillNumber",  "resource": "<http://cargo.iata.org/AirWaybill>",  "message": "AirWaybill number could not be dereferenced, an error occurred"  }] } |

|  |  |  |
| --- | --- | --- |
| Field | **Description** | **Required** |
| @id | A unique identifier of the error on the ONE Record Server. | YES |
| @type | Error model from ONE Record ontology. | YES |
| title | A short, human-readable summary of the problem that SHOULD NOT change from occurrence to occurrence of the problem, except for purposes of localization. | YES |
| code | A ONE Record application-specific error code expressed as a string value. | NO |
| attribute | Field which was not validated correctly / generated the error. | NO |
| resource | Schema/Class that contains the non-validated element. | NO |
| message | A human-readable explanation specific to this occurrence of the problem. Like *title*, this field’s value can be localized. | NO |

There is a list of non-exhaustive JSON-LD syntax error types (relative to 400 Bad Request error family) on the official [JSON-LD API specifications website](https://www.w3.org/TR/json-ld-api/#error-handling).

# ONE Record Security

ONE Record as a basis for data sharing in the Internet of Logistics needs to ensure that data sharing is secure, i.e. that the participants sharing data are known, identified, authenticated and authorized for data access.

Since the Internet of logistics is potentially vast and may cover many different stakeholder groups, there is a need for a network of Identity and Authentication Providers (IAP) that can ensure the validity of the Internet of Logistics participants for their respective stakeholder groups. Each IAP will also hold an inventory of public keys of other IAP’s that they trust. Therefore, Internet of Logistics participants only need to interact with their own IAP and still be able to verify the validity of other participants even though they may be registered with another IAP.

The IAP’s will use Public Key Cryptography to guarantee the authenticity of the Internet of Logistics participant whose identity and roles are in the payload of a signed Json Web Token. Once, validated, this is then used as a bearer token to access the Logistics Object.

## Identity and Authentication Providers (IAP)

Within the Internet of Logistics (IoL) there will be a need for trusted Identity and Authentication Providers (IAP). An example of an IAP could be IATA or other industry entities that represent a group of stakeholders in logistics such as shippers and forwarders but also geographical groupings.

Identity and Authentication Provider

(e.g. IATA)

ONE Record Client

ONE Record Server

token

(2) authenticate

(1) get public keys

token

(3) get LO

Access Rights Ledger of this server

(4) check authorization

(5) serve LO

roles

identities

public keys

Other IAPs

Each IAP would control which other IAPs that they trust.   E.g IATA could trust FIATA or the IRU as an IAP and choose not to trust the state of Nutopia or even Ladonia.

Each IAP would have their own public/private key pair that they would use for signing tokens that would be provided to their members during authentication.

Each IAP would maintain the list of public keys of other IAPs that they trust. e.g IATA would have a record of FIATA's public key and any other IAP's that they trust. This list of IAP public keys would be available to its members via an API.

|  |
| --- |
| //An IAP would provide an API that would return a list of public keys that they trust. The API to retrieve the list of public keys would return a JSON Web Key set as per <https://tools.ietf.org/html/rfc7517> e.g:     {"keys":        [          {           "kty":"RSA",           "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc\_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ\_2W-5JsGY4Hc5n9yBXArwl93lqt7\_RN5w6Cf0h4QyQ5v-65YGjQR0\_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbISD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G\_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",           "e":"AQAB",           "alg":"RS256",           "use": "sig",           "key\_ops": "verify",           "kid":"dcf1"          },          {           "kty":"RSA",           "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc\_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ\_2W-5JsGY4Hc5n9yBXArwl93lqt7\_RN5w6Cf0h4QyQ5v-65YGjQR0\_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbISD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G\_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",           "e":"AQAB",           "alg":"RS256",           "use": "sig",           "key\_ops": "verify",           "kid":"iata1"          }        ] } |

## Authentication

OAuth 2.0 using JWT provides the foundation for authentication within the Internet of Logistics.

IoL participants MUST authenticate against an IAP.

IoL participants would receive a JWT when authenticating against an IAP.

The JWT would be signed using the IAP's private key and would include the id (in the kid property) of the public key in the JWT header (JOSE header).

### JWT Access Tokens from an IAP

The JWT Access Tokens have a header, payload and signature.

### JOSE Header

|  |
| --- |
| {   "alg": "RS256",   "typ": "JWT",   "kid": "dcf1" } |

### Payload

|  |
| --- |
| {     "iss": "http://onerecord.iata.org", // the IAP     "sub": "widgetco", // the user     "exp": 1541859828, //The expiration time of the JWT     "iat": 1516239022, // The time at which the JWT was issued. e.g.      "jti": "dce6023b-375f-4a35-9b4a-41128bf616f" // the id of the JWT      "[https://github./schema/iolIdentifier"](https://github./schema/iolIdentifier%22): "[https://widgetco.net/lelmeeab"](https://widgetco.net/lelmeeab%22), // The IOL identifier of the server the subject belongs to     "[https://github.com.org/IATA-Cargo/ONE-Record/schema/role"](https://github.com.org/IATA-Cargo/ONE-Record/schema/role%22): "SHP" } |

### Signature

The final part of the JWT is the signature using the private key of the IAP that is providing the token.

### Token leakage

In general, an OAUTH 2.0 solution must avoid the leakage of tokens as much as possible as the tokens give access to resources. However, in the Internet of Logistics tokens will be leaked as part of normal processing. Therefore, it will be important that the JWT access tokens are bound to the sender to avoid the possibility of impersonation.

#### Options to solve the problem

To bind the token to the user so that only the user who requested the token can use it. There are several options to achieve this according to [IETF OAUTH 2.0 Best Practices](https://tools.ietf.org/id/draft-ietf-oauth-security-topics-05.html#rfc.section.3.7.1.2):

* [Oauth Token Binding](https://tools.ietf.org/html/draft-ietf-oauth-token-binding-06)
  + **Con**: Google has not adopted it. Technically challenging. Considerable overhead with federated model.
  + **Pro**: Uses Token Binding key pair from TLS connection to prove ownership of Token. Very secure, prevents man in the middle and replay attacks.
* [OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens](https://tools.ietf.org/html/draft-ietf-oauth-mtls-07)
  + **Con**: Enforces the use of mutual TLS. Users need to present a certificate when interacting with a resource provider.
  + **Pro**: Mutual TLS is secure, prevents man in the middle and replay attacks
* [A Method for signing HTTP Requests for Oauth](https://tools.ietf.org/html/draft-ietf-oauth-signed-http-request-03)
  + **Con**: less secure than previous two options, does not prevent man in the middle attack but would limit its impact. Implementation robustness is a risk.
  + **Pro**: Easier to implement. More flexibility. Ability to sign POST contents as well - data integrity and non-repudiation benefits.
* [The OAuth 2.0 Authorization Framework: JWT Pop Token Usage](https://tools.ietf.org/html/draft-sakimura-oauth-jpop-04)
  + **Con**: Similar to previous. Would need to define the nounce or how to create it. Not out of the box support of POST content signature. Implementation robustness is a risk.
  + **Pro**: Easiest to implement if we don't implement server provided nounces.

#### Recommendation

To use PoP (Proof of Possession) tokens instead of Bearer tokens as described in – [A Method for Signing HTTP Requests for OAuth](https://tools.ietf.org/html/draft-ietf-oauth-signed-http-request-03). Promote standard libraries in Logistics community to mitigate implementation robustness risks.

[Oauth Token Binding](https://tools.ietf.org/html/draft-ietf-oauth-token-binding-06) is technically challenging and could prevent adoption, the federated model is not ideal as new Access tokens need to be requested per LDI / One Record Server that the client interacts with. [OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens](https://tools.ietf.org/html/draft-ietf-oauth-mtls-07) requires all users to present client certificates which results in a lot of friction. The [OAuth 2.0 Authorization Framework: JWT Pop Token Usage](https://tools.ietf.org/html/draft-sakimura-oauth-jpop-04) is similar to [A Method for Signing HTTP Requests for Oauth](https://tools.ietf.org/html/draft-ietf-oauth-signed-http-request-03), however relies on server provider nounces which is a challenge and does not bring the data integrity and non-repudiation benefits.

PoP tokens can be created on the fly by the client and can only be used for the resource identified in the token. Therefore, a receiving resource provider cannot reuse the token. Similarly, the timestamp means that servers can make decisions to accept or reject requests based on their age and avoid replay attacks. Signatures of the requests provide data integrity and non-repudiation benefits.

Below is a summary of the solution:

Pre-step

1. User is registered in an IAP.
2. Clients are also registered in an IAP (with a public key).

During authentication

1. A registered user will authenticate with an IAP via a registered client (or with M2M only the registered client will authenticate with an IAP) and receive an access token with following characteristics:
2. Includes user and company user (or client) belongs to in token.
3. Includes the clients public key identifier as registered in the IAP.
4. Signed by the IAP.
5. Consumer requests an LDI/One Record service and includes in the request:
6. PoP token which has the following characteristics:
7. Includes the access token.
8. Includes the resource that is being requested.
9. Time and date of request.
10. Signed by client with the matching private key of the public key registered for that client (and included in the access token).
11. LDI/One Record service upon receiving a request
12. Validates the Access token.
13. Validates that the PoP token matches the resource requested and the public key used to sign the PoP token matches the public key in the Access token.
14. Can in addition validate that the time and date of request is within a threshold.

## Token Verification

When a ONE Record Server receives a request from another IoL participant with a JWT they would need to first verify the JWT before accepting the request. The verification of the JWT would include:

Pre-Step - As a pre-requisite the IoL participant would download and cache the list of public keys of trusted IAP's from their IAP.  E.g an airline would download the list of public keys of trusted IAP's from IATA. IATA would also maintain the list of public keys of other accredited IAPs that it trusts.  The cache would be refreshed on a periodic basis and could also refresh on a trigger such as receiving a signature with an id that is not in the cache.

Verification Step - When an IoL participant receives a request they would verify the JWT to ensure:

* That it is valid (not expired using exp property)
* It is signed by an IAP that is trusted by their provider (using the kid property to identify which public key to use to verify the signature).

## Authorization

The two aspects to authorization for a ONE Record Server are:

* Does the authenticated requestor have access to the LO?  See Authorization to a Logistics Object below.
* If yes, then what data does the requestor have access to within the LO?  See Field level authorization within a Logistics Object.

### Authorization to a Logistics Object

Authorization to a Logistics Object can be given in two ways:

* Explicitly in the Logistics Object - A company can be given access to a LO by specifying the Server Identifier in the Logistics Object.
* Using the ONE Record Server to PATCH the Logistics Object with additional partner access. Any company that has access to the LO can cascade the trust to other companies within the IoL.

### Field level authorization within a Logistics Object

Field level authorization within a Logistics Object is possible using standardized roles.  Based on the user role the ONE Record Server could decide to only provide certain field elements within the Logistics Object to the requestor.

# Publish & Subscribe with ONE Record

It is important that companies can receive data into their backend systems in a near real time manner and ONE Record proposes a Publish & Subscribe pattern to allow for a distributed network of ONE Record compliant platforms.

This document describes the Publish & Subscribe model, its implementation and the requirements of a Client Subscription API which a company must implement to receive Logistics Objects from ONE Record Servers through subscriptions.

ONE Record Client

ONE Record Server

API

API

Subscriber Publisher

## Publish & Subscribe model

### Publish/subscribe topics and guaranteed delivery queue

Data is exchanged between applications using a notion of topics and delivery queues. While in transit, data is kept in message queues that ensure integrity and availability of the system. Should a subscribing application go down, messages are safely retained until the recipient is ready to read them again.

For each subscriber and each topic, a message queue is maintained automatically by the publisher to keep data in until the subscriber confirms it has received a particular object.

Two scenarios were identified for initiating the publish/subscribe process. For simplicity reasons, the security part was not detailed in the following diagrams.

### Scenario 1 – Publisher/Subscriber initiated by the Publisher

In the first scenario, the subscription process is initiated by the publisher.

The following steps describe how publish and subscribe is proposed to be implemented in the Internet of Logistics (*webhook model with dynamic subscription*):

****Step 1 - Publish a Logistics Object****

The publish action occurs when a Logistics Object is created one a ONE Record Server. At this stage the Logistics Object is accessible via the Server API to authorized companies.

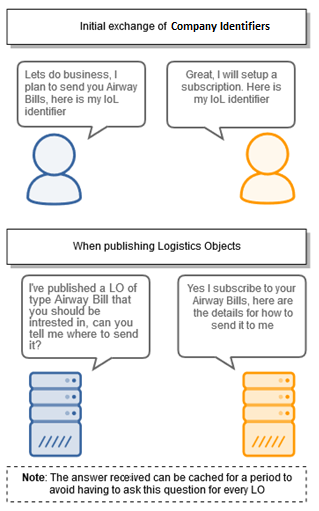
****Step 2 - Retrieve Subscription information from companies that you want to give access to****

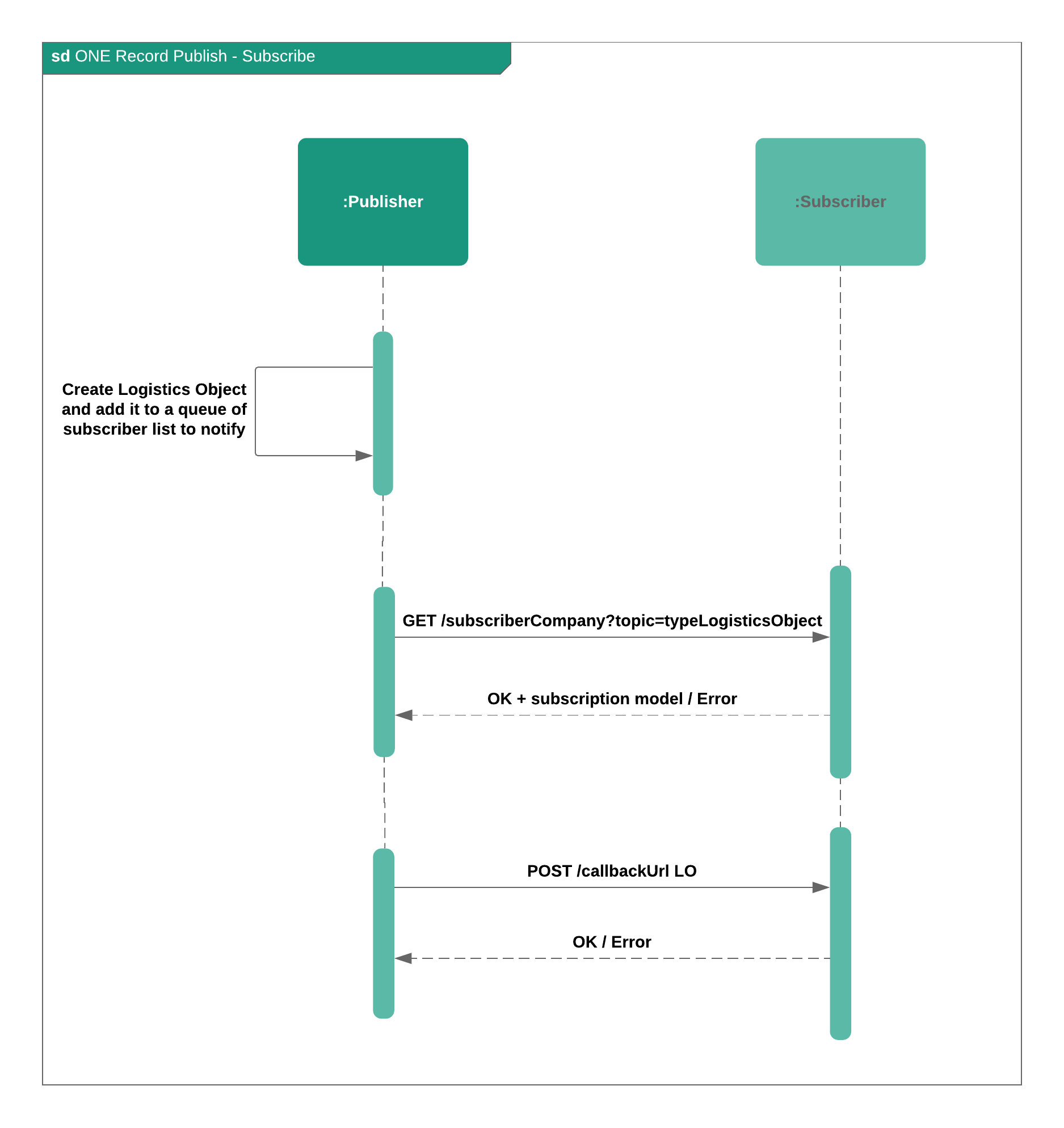
The second step is retrieving the subscription information from the companies you want to give access to this Logistics Object. To achieve this, the company publishing the Logistics Object must check with each of the companies it wants to give access to, whether they subscribe to these types of Logistics Objects. If they do, they provide the details of the endpoint where the Logistics Objects should be pushed to.

The prerequisite to this is that the companies must know each other through a previous exchanged Company Identifier so that the machines can ask this question during operation. These Company Identifiers may also be retrieved form common or local directories.

****Step 3 - Push to the company’s ONE Record Clients****

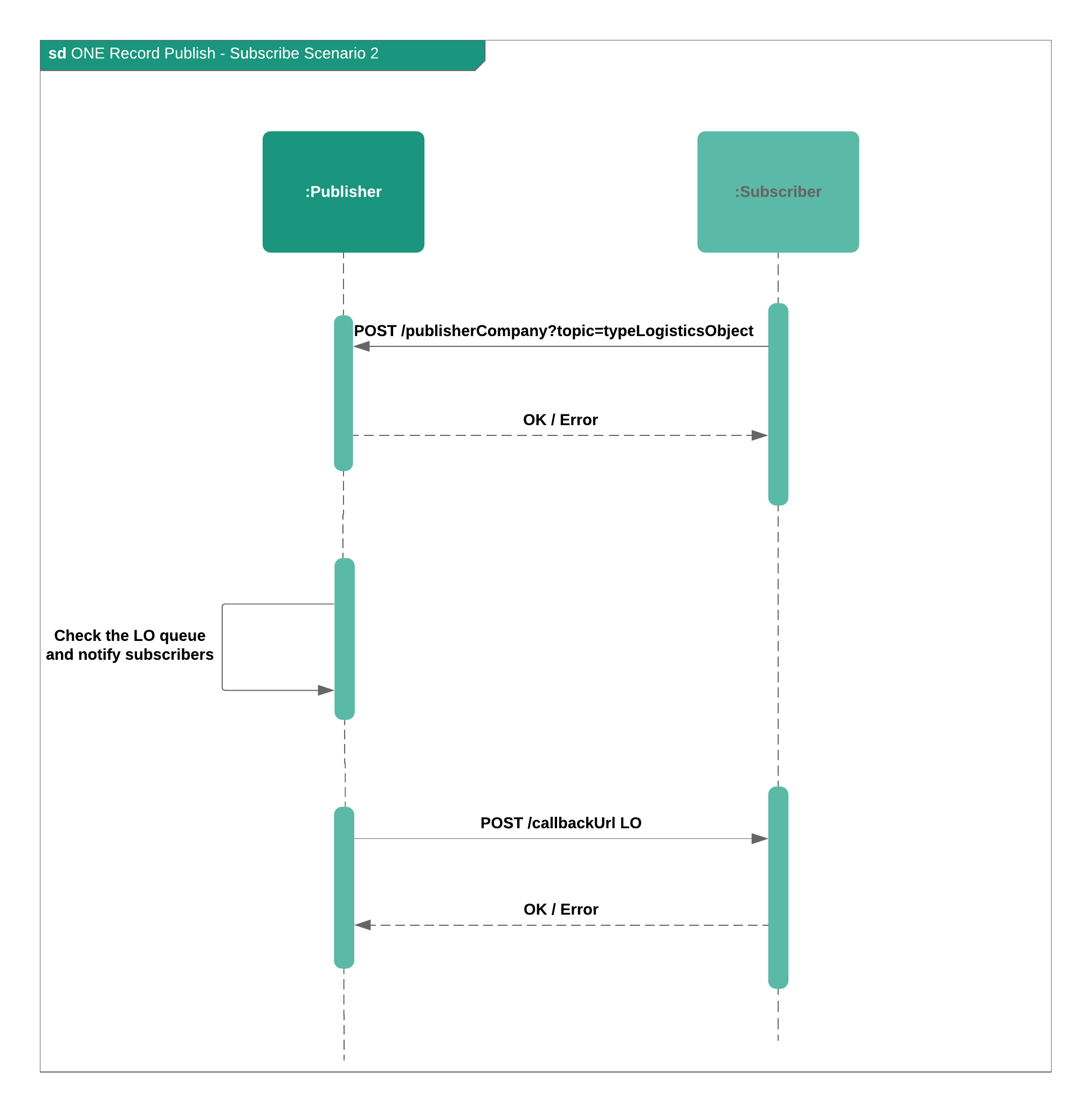
Once the subscription information is received the publisher would push the Logistics Object to the intended ONE Record Client using the details provided. If Client Subscription API (server) was not available at the time, then the publisher would need to queue and retry to publish the Logistics Object over a certain time.

`



### Scenario 2 – Publisher/Subscriber initiated by the Subscriber TODO

In the second scenario, the subscriber initiates the subscription process by pulling the publisher in order to verify if there are any logistics objects/updates of a given topic to which it can subscribe to.



## The ONE Record Company Identifier

The ONE Record Company Identifier is a unique identifier for a company in the internet of logistics. It must be a URL that is unique to a specific company. An example of a Company Identifier is given below:

https:// {Server Domain} / {license plate}

However, it is **more** than just an identifier, it is also behaving as an address for the server that can be used to retrieve company and subscription information and should be stored in publisher’s backend system.

The characteristics that make a URL a Company Identifier are the following:

* **Company Information** - If you perform an authorized GET request on the URL it will return basic company information;
* Subscription Information - If you perform a GET request on the URL with a topic query parameter it will return subscription information if the server subscribes to the topic (i.e. type of Logistics Object) from your server.

The Company Identifier is used in the following use cases:

* **In a Bearer Token** - It is included in the Bearer Token so that the company the user belongs to can identified;
* **In a Logistics Objects** – It is included in Logistics Objects to identify companies related to shipment.
* **For Authorization** - Included when giving companies access to Logistics Objects.

Note: Authorization to a Logistics Object can be implicitly given by including the Company Identifier in the Logistics Object or it can be explicitly given by using the PATCH request on the ORS-API.

## Get Company Information from the IoL

Retrieves basic company information.

### Request

HTTP Request type: GET

GET /CompanyB

Host: myonerecordserver.net

Authorization: mybearertokenbase64encoded

Accept: application/ld+json

### HTTP Headers

The following HTTP header parameters MUST be present in the GET company information request:

|  |  |
| --- | --- |
| Authorization | A valid Bearer Token |
| Accept | The content type that you want the HTTP response to be formatted in. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |

### Response

{

"@context": {

"@vocab" : "http://cargo.iata.org/"

},

"@id": "https://myonerecordserver.org/my\_license\_plate",

"@type": "CompanyInformation",

"company": {

"@type": "Company",

"name": "text",

"IATACargoAgentCode": "numeric",

"branch": [{

"branchName": "text",

"IATACargoAgentLocationIdentifier": "numeric",

"otherIdentifier": [{

"identifierName": "text",

"identifer": "text"

}],

"location": {

"@type": "Location",

"type": "text",

"code": "text",

"name": "text",

"geoLocation": {

"@type": "GeoLocation",

"latitude": {

"@type": "Value",

"value": "text",

"unit": "text"

},

"longitude": {

"@type": "Value",

"value": "text",

"unit": "text"

},

"elevation": {

"@type": "Value",

"value": "text",

"unit": "text"

}

},

"address": {

"@type": "Address",

"street": "text",

"POBox": "text",

"postalCode": "text",

"cityCode": "text",

"cityName": "text",

"regionCode": "text",

"regionName": "text",

"countryCode": "text",

"countryName": "text",

"addressCodeType": "text",

"addressCode": "text"

}

},

"contactPerson": {

"@type": "Person",

"contactType": "text",

"salutation": "text",

"firstName": "text",

"middleName": "text",

"lastName": "text",

"jobTitle": "text",

"department": "text",

"employeeID": "text",

"contactDetails": [{

"phoneNumber": "text",

"emailAddress": "text",

"other": {

"type": "text",

"detail": "text"

}

}]

}

}],

"airlineCode": "text",

"airlinePrefix": "text"

},

"supportedLogisticsObjects": [

"http://cargo.iata.org/HouseManifest",

"http://cargo.iata.org/Airwaybill"

],

"supportedContentTypes": [

"application/ld+json",

"application/x-turtle",

"text/turtle"],

"serverEndpoint": "https://myonerecordserver.org/"

}

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 200 | Request for company information is successful | CompanyIdentifier |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to retrieve company information | Error model |
| 404 | Not found | Error model |

## Get Subscription Information

Retrieves subscription information if the server subscribes to a topic:

### Request

HTTP Request type: GET

GET /CompanyB?topic=AirwayBill

Host: myonerecordserver.net

Authorization: mybearertokenbase64encoded

Accept: application/ld+json

### HTTP Headers

The following HTTP header parameters MUST be present in the GET subscription information request:

|  |  |
| --- | --- |
| Authorization | A valid Bearer Token |
| Accept | The content type that you want the HTTP response to be formatted in. Valid content types include:   * **application/x-turtle or text/turtle** * **application/ld+json** |

### Response

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 200 | Request for subscription information is successful | Subscription |
| 204 | Request has been successful, but the server does not subscribe | No response body |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to retrieve subscription information | Error model |
| 404 | Not Found | Error model |

## Subscriptions

In order to receive Logistics Objects automatically you must setup a subscription in the relevant ONE Record Server.  The subscription information includes:

1. The Company Identifier of the company you want to subscribe to (delegation scenario). (*subscribedTo*)
2. The Logistics Object type that you want to subscribe to. (*topic*)
3. The callback URL of the Client Subscription API where you want to receive Logistics Objects. (Refer requirements of your endpoint below.) (*callbackUrl*)
4. The content type you want to receive. (*contentType*)
5. Either a secret or API Key that ensures that only companies with this subscription information can post to your endpoint. (*secret*)
6. You must also specify if you want to receive updates on that Logistics Object. (*subscribeToStatusUpdates*)
7. Flag specifying if the publisher should send the whole logistics object or not in the notification object. (*sendLogisticsObjectBody*)
8. Duration of the period to cache the subscription information in seconds. (*cacheFor*)
9. A subscriber could also send specific field filter to which it wants to subscribe to. (e.g. destination countries). TODO

### Subscription response model

Example of subscription response model:

{

"@context": {

"@vocab": "http://cargo.iata.org/”

},

"@id": "https://subscriberonerecordserver?topic=ShippersInstructions",

"@type": "Subscription",

"subscribedTo": "https://publisheronerecordserver.net/yourCompany",

"topic": <http://cargo.iata.org/AirwayBill>,

"callbackUrl": "<https://subscriberonerecordserver/callback>",

"contentType": ["application/ld+json"],

"secret": "C89583BA9B1FEEAB25F715A3BA2F3",

"subscribeToStatusUpdates": "true",

"sendLogisticsObjectBody": "true",

"cacheFor": "86400"

}

## Notification Request

The publisher sends a notification request to the subscriber when a logistics object is created or updated. If the subscriber chose to receive the entire logistics object body via *sendLogisticsObjectBody=true* field, then the whole object is sent.

### Request body for notification without the content of the object:

{

"@context": {

        "@vocab": "http://cargo.iata.org/"

    },

    "@type": "Notification",

    "eventType": "OBJECT\_CREATED",

    "topic": "http://cargo.iata.org/AirwayBill",

    "logisticsObjectRef": "<https://server/licence_plate/s>ome\_id"

}

Where:

1. EventType: OBJECT\_CREATED or OBJECT\_UPDATED.
2. Topic: The type of Logistics Object.
3. LogisticsObjectRef: The Logistics Object for which the notification is sent.

## ONE Record Client Subscription API

The Client Subscription API is required to receive data from ONE Record Servers via a Subscription. Unlike the Server API, which can be accessed by any Internet of Logistics participant with adequate rights, the Client Subscription API is only exposed to ONE Record Servers with whom the company has set up a Subscription to agreed Logistics Objects.

The Client Subscription API:

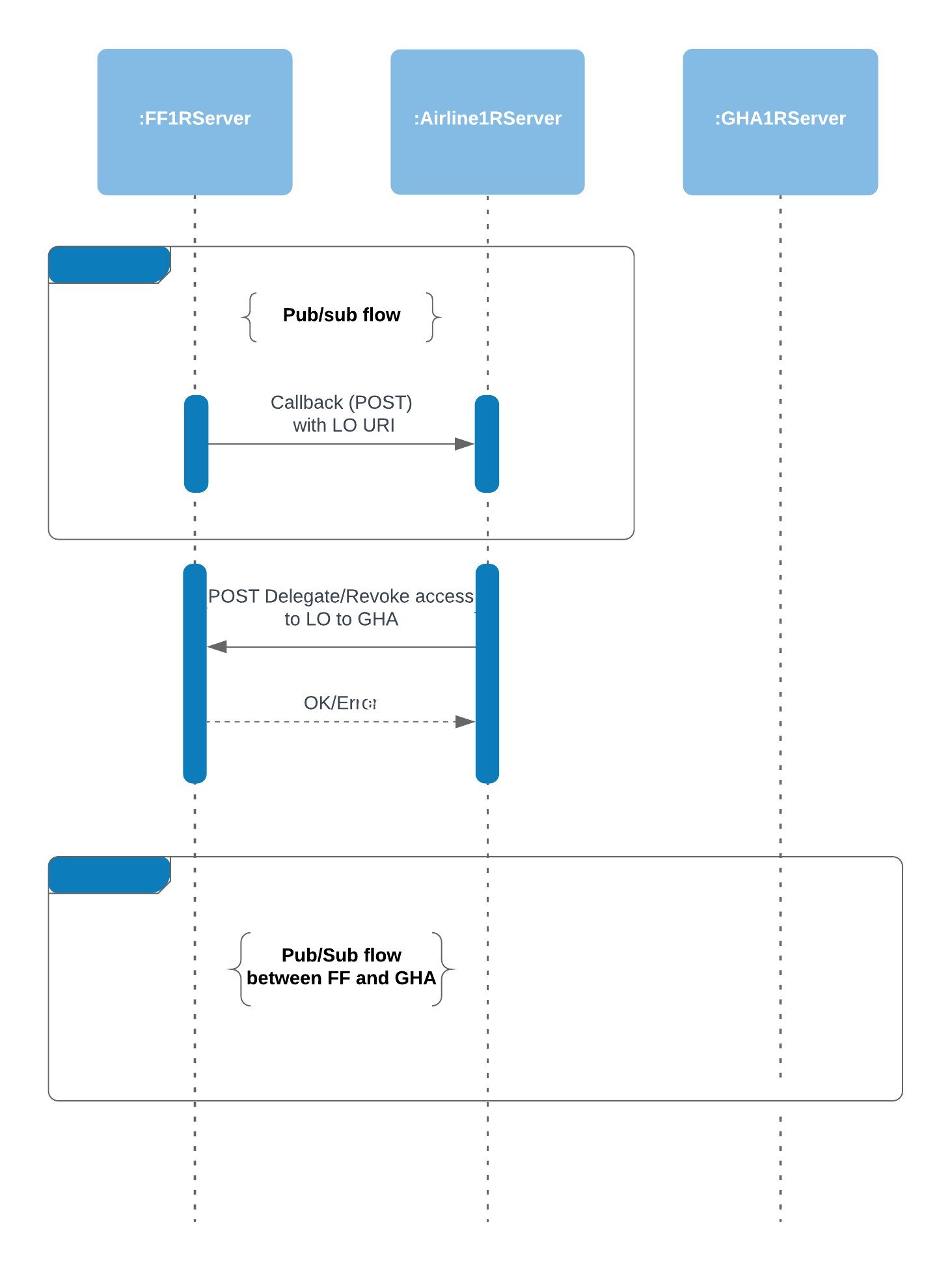
* MUST support HTTP 1.1
* MUST support TLS 1.2
* MUST support the POST request on the endpoint.
* MUST expect a Logistics Object in the POST request. Note only the content types that you specify in the subscription request need to be supported.
* MUST respond with a 2XX response when it receives the Logistics Object.
* MUST verify either the HMAC signature or API key to ensure only authorized requests are processed.
  + The HMAC (<https://tools.ietf.org/html/rfc6151>) signature (in the header property X-Hub-Signature) with a shared subscription secret can be used to authorise the request. If the signature does not match, subscribers must locally ignore the message as invalid. Subscribers may still acknowledge this request with a 2xx response code in order to be able to process the message asynchronously and/or prevent brute-force attempts of the signature.
  + The API key (x-api-key) can also be used to authorize requests to the subscription endpoint.

The Client Subscription API can also expect to receive the following HTTP Headers:

|  |  |  |
| --- | --- | --- |
| **Header** | **Description** | **Expected** |
| URI-resource | The top-level URI of the Logistics Object of the resource being sent to the Client Subscriber.  Note: As RDF can come in any order of triples it is not always trivial to determine the top-level resource ID. This header will inform the ORC of the top-level resource URI | Yes, always |
| Resource-Type | Class of the logistics object sent (e.g. AirwayBill, Booking) | Yes, always |
| Orig-Request-Method | The HTTP request method that was used that generated this message to the ORC. Values here are the typical HTTP/REST verbs e.g. POST, PATCH | Yes, always |
| X-Hub-Signature | If a secret has been provided in the subscription, a HMAC signature would be present in this header | Optional, dependant on Subscription |
| x-api-key | If an API key is provided in the subscription, the API key would be provided in this header | Optional, dependant on subscription |
| Authorization | The ONE Record Access token of the ONE Record Server - to be discussed. This would require sender binding to make sense. There is also a discussion in that the owner of this API is not owning the security which may be an issue. The API Key or HMAC signature in contrast would be based on security information provided by the owner of the endpoint. | To be discussed. |

# Delegation

In ONE Record parties are enabled to grant other parties access to (parts of) their data. The standard allows parties to modify or withdraw these access rights to their data, whenever they wish.

The party granting access is referred to as the *delegator* and the party receiving the access is the *delegate*. 

### Request

HTTP Request type: POST

POST /delegation

Host: myonerecordserver.net

Authorization: mybearertokenbase64encoded

Accept: application/ld+json

### Delegation model

|  |
| --- |
| {  "@context": {  "@vocab": "http://cargo.iata.org/"  },  "@type": "DelegationRequest",  "targetLogisticsObject":"https://1recordserver.org/my\_airline/airwaybill\_123",  "targetCompany": "https:/onerecordserver.org/delegated\_company\_identifier",  "action": "DELEGATE" // or REVOKE  } |

|  |  |  |
| --- | --- | --- |
| Field | **Description** | **Required** |
| @type | DelegationRequest | YES |
| targetLogisticsObject | The identifier of the logistics object to which the access is requested | YES |
| targetCompany | The party that receives the delegated rights | YES |
| action | DELEGATE or REVOKE | YES |

### Response

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 204 | Request for delegation was successful | No response body |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to retrieve delegate | Error model |
| 404 | Not Found | Error model |

# Events (status update) in ONE Record

Status updates in ONE Record can pe added to Logistics Objects through Events. By definition, each Logistics Object can be assigned events.

## Create event

### Request

HTTP Request type: POST

POST logisticsObjectURI/events

Host: myonerecordserver.net

Authorization: mybearertokenbase64encoded

Accept: application/ld+json

Example:

**POST** https://www.onerecordcargo.org/my\_airline/shipment\_123456/events

### Request body:

|  |
| --- |
| {  "@context": {  "@vocab": "http://cargo.iata.org/"  },  "@type": "Event",  "logisticsObjectRef": "https://1recordserver.org/my\_airline/shipment\_123456",  "statusBy": "https:/1recordserver.org/some\_company\_identifier",  "status": "DEP",  "statusTimestamp": "2020-03-09T14:05:17+00:00",  "statusApplicableTo": ["https://1recordserver.org/my\_airline/booking\_123456"],  "geo": {  "@type": "Location",  "code": "text",  "name": "text",  "geolocation": {  "@type": "Geolocation",  "latitude": "text",  "longitude": "text",  "elevation": "text"  },  "address": {  "@type": "Address",  "street": "text",  "poBox": "text",  "cityCode": "text",  "cityName": "text",  "regionCode": "text",  "regionName": "text",  "addressCodeType": "text",  "addressCode": "text",  "country": {  "@type": "Country",  "countryCode": "text",  "countryName": "text"  }  }  }  } |

|  |  |  |
| --- | --- | --- |
| Field | **Description** | **Required** |
| @type | Event | YES |
| logisticsObjectRef | Logistics object the event is valid for | YES |
| statusBy | IRI, company IoL identifier | YES |
| status | Valid status | YES |
| statusTimestamp | Date and time when the event occurred | YES |
| statusApplicableTo | IRI, used only when the event is only applicable to certain linked logistics objects in the primary logistics object (logisticsObjectRef) | NO |
| geo | Geo position of where the event occurred (of type ONE Record - Location) | NO |

### Response

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 201 | Event has been published to the Internet of Logistics. | No body required |
| 400 | Invalid Event | Error model |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to publish an event update to the Internet of Logistics | Error model |
| 415 | Unsupported Content Type | Error model |

## Retrieve events

### Request

HTTP Request type: GET

GET logisticsObjectURI/events

Host: myonerecordserver.net

Authorization: mybearertokenbase64encoded

Accept: application/ld+json

Example:

**GET** <https://www.onerecordcargo.org/my_airline/shipment_123456/events>

### Response

A positive HTTP 200 response is expected to a GET request. The body of the response is expected to be the events list in the format that has been requested in the Accept header of the request.

|  |  |  |
| --- | --- | --- |
| Code | Description | Response Body |
| 200 | The request to retrieve the Event list has been successful | Event |
| 401 | Not authenticated or expired token | Error model |
| 403 | Not authorized to retrieve the Events | Error model |
| 404 | Logistics Object not found | Error model |

# Glossary

|  |  |
| --- | --- |
| **Term** | **Description** |
| Authentication | A process that validates the identity of IoL participant |
| Authorization | A process that determines whether a IoL participant is allowed to access a specific Logistics Object |
| Identity & Authentication Provider | A service that allows Internet of Logistics participants register and obtain an Public Key encrypted token identify themselves with ONE Record Servers and get access to Logistics Objects |
| Internet of Logistics (IoL) | A network of ONE Record Clients and Servers that can share Logistics Objects over the internet using the ONE Record standard data model, APIs and security |
| JSON-LD | JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB. |
| Json Web Token (JWT) | Json specification for a token format that includes a user defined payload and the option for encryption. |
| Linked Data | Linked Data empowers people that publish and use information on the Web. It is a way to create a network of standards-based, machine-readable data across Web sites. It allows an application to start at one piece of Linked Data and follow embedded links to other pieces of Linked Data that are hosted on different sites across the Web. |
| Logistics Object | A data object that represents a meaningful entity in the logistics business. These may represent documents like air waybills but may also be more granular such as company details or a transport segment description. Logistics Objects are specified in a common data model by IATA and transport and logistics partners. |
| OAUTH2 | A protocol for delegation of authentication in a network of secure systems |
| ONE Record Client | A system that can access Logistics Objects on a ONE Record Server. This system may also have a ONE Record Subscriber API. |
| ONE Record Server | The platform that hosts Logistics Objects on a web server on behalf of one or more companies |
| ONE Record Subscriber API | A ONE Record Client API that has dedicated endpoint(s) for receiving Logistics Objects via a subscription |
| Participant | Server that access or shares data via the Internet of Logistics and that has registered with an Accredited Identity Provider and has possession of a valid certificate to prove this |
| Public Key Cryptography | An encryption technology that uses public and private keys to guarantee the authenticity of encrypted data without the need for both parties to share a common secret |
| Publisher | The Party that makes their Logistics Objects available through a ONE Record Server |
| Subscriber | The Party that subscribes to Logistics Objects in order to receive updates automatically |
| URI | In the web context, this is a URL that uniquely identifies a Logistics Object and a Host |