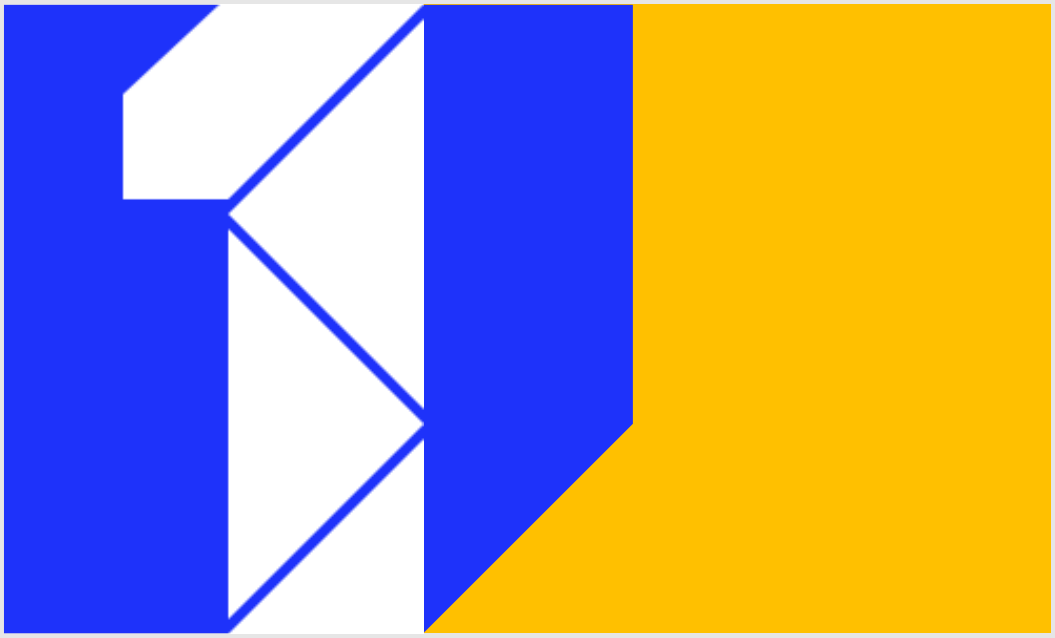
ONE Record API & Security

Reference Specification



Contents

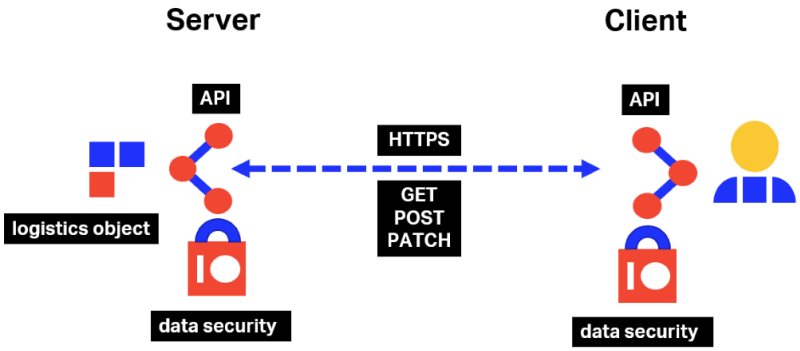
Change log

Change No.	Description	Modified by	Date Submitted	Comments
1	Add scenario 2 Add scenario 2 for Publisher/Subscriber (process initiated by the subscriber)	A. Blaj	2019/10/16	
2	Add delegation model	A. Blaj	2019/10/16	
3	Rename serverInformation to companyInformation. Add companyIdentifier	A. Blaj	2019/11/07	
4	Add notification model for pub/sub	A. Blaj	2019/12/19	
5	Add logisticsObjectRef field to audit trail model	A. Blaj	2020/01/22	
6	Add Events (status updates) specifications	A. Blaj	2020/03/09	
7	Change document layout according to IATA official template	A. Blaj	2020/04/06	
8	Add specifications for Access Control	A. Blaj	2020/04/07	
9	Add Publish & Subscribe in multi-link/multiparty scenario specifications proposal	A. Blaj	2020/04/09	
10	Add second use case for delegation	A. Blaj	2020/04/09	

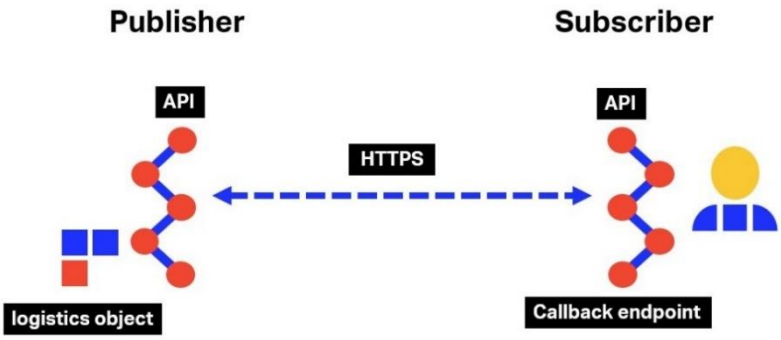
Overview

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.



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

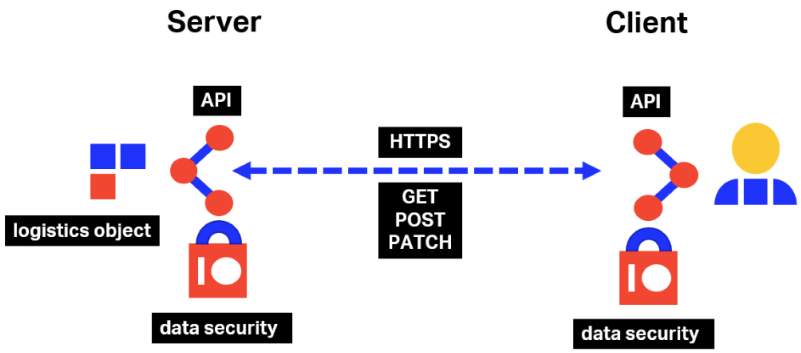


This document is in several parts: the first part specifies the ONE Record Server API, the second one includes the publisher/subscriber model, the third part presents the delegation in ONE Record, the fourth one the events creation/update, the fifth one the Access Control and the last part concerns the ONE Record security specifications.

ONE Record 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



Logistics Object ID

Every Logistics Object is identified by a **Logistics Objects ID**. A Logistics Object ID (LOID) can be any URL which is unique. An example of a LOID could be for example:

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

Field	Description
ORS Domain	The domain name associated with the ONE Record Server e.g. `onerecordcargo.org`
License plate	The company identifier for this ONE Record Server, e.g. `myAirline`
Unique ID to identify LO	An identifier for the Logistics Object that is unique at least for this server. An example of a LOID is: `https://onerecordcargo.org/myAirline/airwaybill_123-12345678` The LOID should be URL friendly, i.e. avoid unsafe characters that include the blank/empty spaces and " < > # % { } | \ ^ ~ [ ] `.

Create Logistics Object (POST)

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 LOID.

Request

HTTP Request type: **POST**

HTTP Headers

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

Header	Description
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
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 in the ONE Record ontology: https://github.com/IATA-Cargo/ONE-Record/

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	Error model
403	Not authorized to publish the Logistics Object to the server	Error model
415	Unsupported Content Type	Error model

Read Logistics Object (GET)

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:

Header	Description
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

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.

`GET` request could also return a `Link` Header with the location of the Access Control List (see Access Control List Resources). If the Logistics Object does not have an individual ACL (and therefore relies on an implicit ACL from a parent), this link header will still be present, but will return a 404.

```
Link: <http://myServer/myAirline/logisticsObject/acl>; rel="acl"
```

Code	Description	Response body
200	The request to retrieve the Logistics Object has been successful	Logistics Object
401	Not authenticated	Error model
403	Not authorized to retrieve the Logistics Object	Error model
404	Logistics Object not found	Error model

Update Logistics Object (PATCH)

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 with `PATCH` in ONE Record are inspired by 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://onerecordcargo.org/myAirline/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.
- 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:

Header	Description
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
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://onerecord.iata.org/AirWaybill#totalPieceAndULDCount",
         "o":{
            "value":"10",
            "datatype":"https://www.w3.org/2001/XMLSchema#decimal"
         }
      },
      {
         "op":"add",
         "p":"http://onerecord.iata.org/AirWaybill#totalPieceAndULDCount",
         "o":{
            "value":"11",
            "datatype":"https://www.w3.org/2001/XMLSchema#decimal"
         }
      },
      {
         "op":"add",
         "p":" http://onerecord.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`".
- To discuss further: Add provenance information? (Suggestion of ontology to use: 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 request body is invalid	Error model
401	Not authenticated	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

Audit trail of Logistics Object

Request

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
```

HTTP Headers

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

Header	Description
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


```
{
   "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://onerecord.iata.org/AirWaybill#totalPieceAndULDCount"
,
                  "o":{
                     "value":"10",
                     "datatype":"https://www.w3.org/2001/XMLSchema#decimal"
                  }
               },
               {
                  "op":"add",
                  "p":"http://onerecord.iata.org/AirWaybill#totalPieceAndULDCount
",
                  "o":{
                     "value":"11",
                     "datatype":"https://www.w3.org/2001/XMLSchema#decimal"
                  }
               },
               {
                  "op":"add",
                  "p":" http://onerecord.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

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://onerecord.iata.org/"
  },
  "@type": "Error",
  "title": "Request contains invalid field",
  "details": [{
    "code": "1234",
    "attribute": ".AirWayBillNumber",
    "resource": "http://onerecord.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.

Publish & Subscribe with ONE Record

ONE Record proposes a Publish & Subscribe pattern to allow for a distributed network of ONE Record compliant platforms.

This chapter 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.

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 and most usual use case, the subscription process is initiated by the publisher

The following steps describe how publish and subscribe is proposed to be implemented in the ONE Record 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

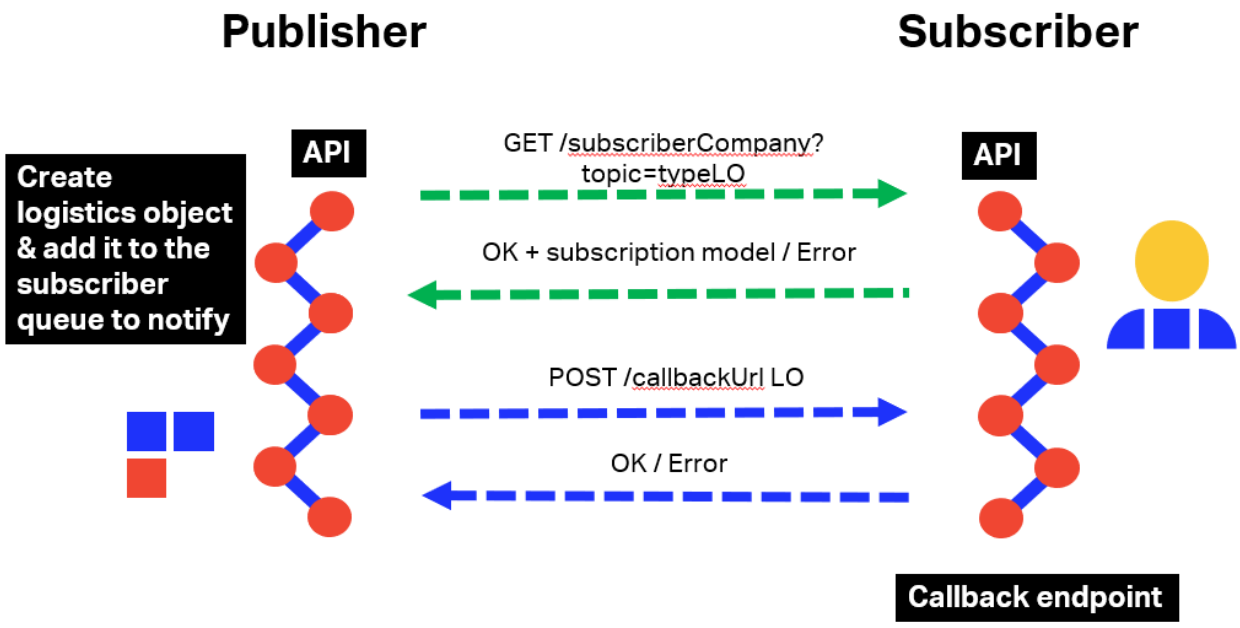
The second step is retrieving the subscription information from the companies you want to give access to this Logistics Object. To achieve this, the company publishing the Logistics Object must check with each of the companies it wants to give access to, whether they subscribe to these types of Logistics 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.

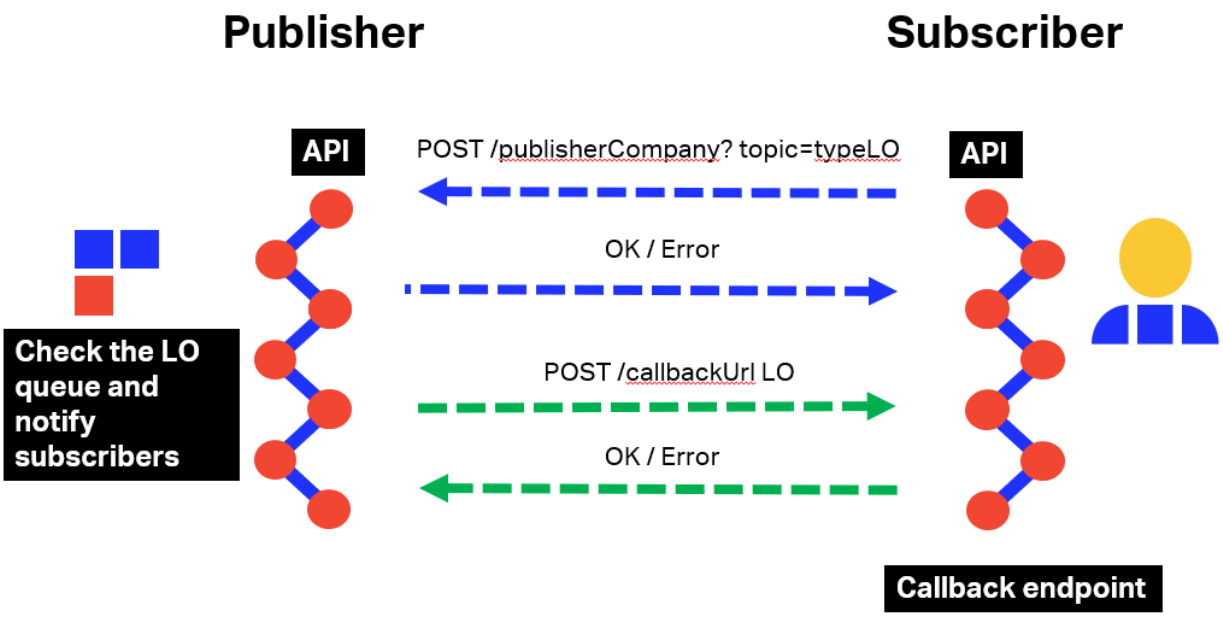
Note: In Publish & Subscribe, publishing parties need to save a list of all the parties subscribed to their Logistics Objects in their backend systems. One of the possibilities would be that the list of subscribers is embedded in the body of the Logistics Object.



Publish & Subscribe Sequence Diagram – Scenario 1

Scenario 2 – Publisher/Subscriber initiated by the Subscriber

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.



Publish & Subscribe Sequence Diagram – Scenario 2

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/Access Control** - 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:

Header	Value
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

```
{
  "@context": {
    "@vocab" : "http://onerecord.iata.org/"
  },
  "@id": "https://myonerecordserver.org/myLicensePlate",
  "@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://onerecord.iata.org/HouseManifest",
     "http://onerecord.iata.org/Airwaybill"
     ],
 "supportedContentTypes": [
     "application/ld+json",
     "application/x-turtle",
```

```
      "text/turtle"],
  "serverEndpoint": "https://myonerecordserver.org/"
}
```

Response

Code	Description	Response body
200	The request to retrieve the Company Information has been successful	Company Identifier
401	Not authenticated	Error model
403	Not authorized to retrieve the Company Information	Error model
404	Company Information 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:

Header	Value
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://onerecord.iata.org/”
    },
    "@id": "https://subscriberonerecordserver?topic=ShippersInstructions",
    "@type": "Subscription",
    "subscribedTo": "https://publisheronerecordserver.net/yourCompany",
    "topic": http://onerecord.iata.org/AirwayBill,
    "callbackUrl": "https://subscriberonerecordserver/callback",
    "contentType": ["application/ld+json"],
    "secret": "C89583BA9B1FEEAB25F715A3BA2F3",
    "subscribeToStatusUpdates": "true",
    "sendLogisticsObjectBody": "true",
    "cacheFor": "86400"
}
```

Where:

Field	Description	Required
subscribedTo	The Company Identifier of the company you want to subscribe to (delegation scenario)	YES
topic	The Logistics Object type that you want to subscribe to	YES
callbackUrl	The callback URL where you want to receive notifications on Logistics Objects. (Refer requirements of your endpoint below)	YES
contentType	The content type you want to receive.	YES
secret	Either a secret or API Key that ensures that only companies with this subscription information can `POST` to your `callbackUrl` endpoint	NO
subscribeToStatusUpdates	You must also specify if you want to receive updates on that Logistics Object	NO
sendLogisticsObjectBody	Flag specifying if the publisher should send the whole logistics object or not in the notification object	YES
cacheFor	Duration of the period to cache the subscription information in seconds	NO

To discuss further: A subscriber could also send specific field filter to which it wants to subscribe to. (e.g. destination countries).

Code	Description	Response body

200	The request to retrieve the Subscription Information has been successful	Subscription
204	Request has been successful, but the server does not subscribe	No response body
401	Not authenticated	Error model
403	Not authorized to retrieve the Subscription Information	Error model
404	Subscription Information not found	Error model

Subscription API Requirements

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	Required
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
Resource-Type	Class of the logistics object sent (e.g. AirwayBill, Booking)	YES

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
X-Hub-Signature	If a secret has been provided in the subscription, a HMAC signature would be present in this header	NO
x-api-key	If an API key is provided in the subscription, the API key would be provided in this header	NO
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 discuss

POST 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

HTTP Request type: **POST**

HTTP Headers

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

Header	Description
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
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

Request body for notification without the content of the object:

```
{
    "@context": {
        "@vocab": "http://onerecord.iata.org/"
    },
    "@type": "Notification",
    "eventType": "OBJECT_CREATED",
    "topic": "http://onerecord.iata.org/AirwayBill",
    "logisticsObjectRef": "https://server/licence_plate/some_id"
}
```

Where:

Field	Description	Required
eventType	`OBJECT_CREATED` or `OBJECT_UPDATED`	YES
topic	Type of Logistics Object	YES
logisticsObjectRef	Logistics Object for which the notification is sent	YES

Response

Code	Description	Response body
204	Notification received successfully	No response body
400	Notification format is invalid	Error model
401	Not authenticated	Error model
403	Not authorized to send a notification request	Error model

Publish & Subscribe in multi-link/multi-party scenario

By definition, Linked Data is interlinked with other data, and the ownership of the different levels of data can be distributed throughout the Internet of Logistics. One of the challenges encountered while building a publisher/subscriber solution for ONE Record is to make sure that when there is an update on a logistics object, all the partners which are subscribed to other data that are linked to this object receive the update.

In order to better illustrate the publisher/subscriber interactions in a multi-party context, we consider a scenario in which a **FirstParty**, a **SecondParty** and a **ThirdParty** are exchanging data in the Internet of Logistics.

Step 1 – FirstParty and SecondParty agree on Data Policy

In ONE Record, every data sharing of any type needs to be supported by data policy. Data policy determines, among others, the rights of the recipients to share data and with whom they do so. This policy determines also what data protection a recipient of data needs to apply. Data policy can have different levels of granularity, depending on the needs of each party.

The process is contractual, and it is effectuated at the beginning of the data exchange partnership between two parties of the Internet of Logistics. The policy can have an expiration date.

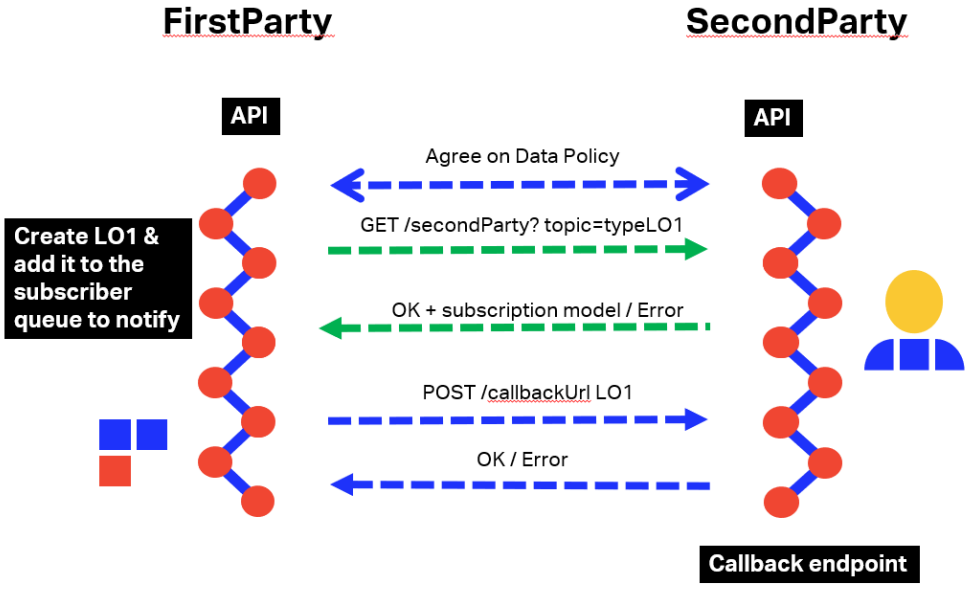
In this first step, FirstParty and SecondParty agree, among others, on which type of data the SecondParty can share to third parties.

Step 2 – FirstParty creates a Logistics Object (LO1) on its ONE Record Server

FirstParty creates a Logistics Object (called LO1 in this document) on its own ONE Record Server, using `POST` specifications of the ONE Record standard.

Step 3 – Publish & Subscribe flow between FirstParty and SecondParty

1. FirstParty asks SecondParty if it wants to subscribe to LO1.
2. SecondParty replies with `OK` and sends its subscription information back to FirstParty. The content of the subscription can be found here.
3. FirstParty sends notification containing the URI of the LO1 on his ONE Record Server to the callback endpoint on the SecondParty ONE Record Server.
4. SecondParty effectuates a `GET` LO1 on the FirstParty ONE Record Server in order to retrieve the full content of the LO1.

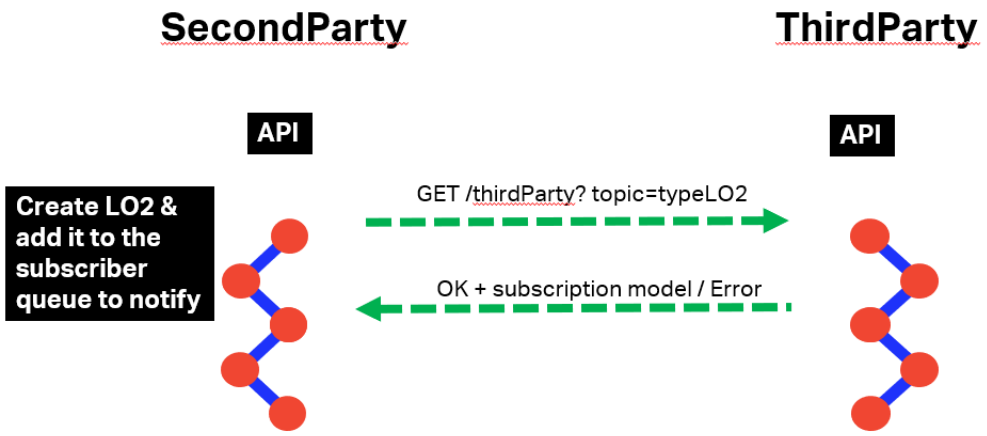


Step 4 – SecondParty creates a Logistics Object (LO2) on its own ONE Record Server

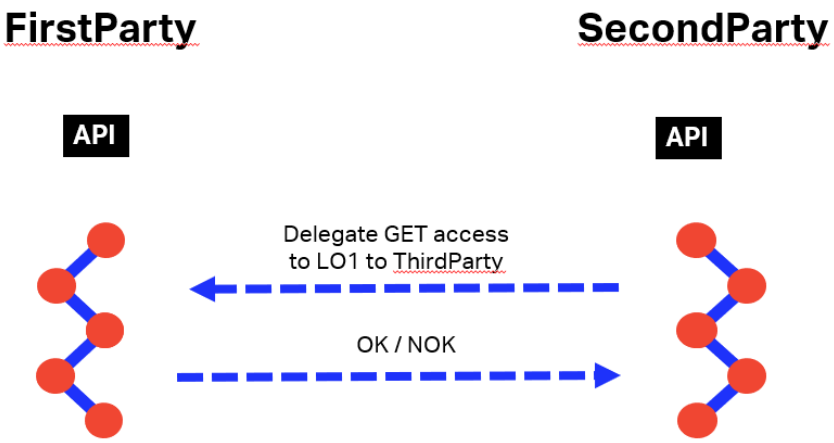
SecondParty creates a Logistics Object (called LO2 in this document) on its own ONE Record Server, using `POST` specifications of the ONE Record standard. **LO2 contains a link to LO1**.

Step 5 – Publish & Subscribe flow between SecondParty and ThirdParty with Delegation of access to LO1 to the ThirdParty

1. SecondParty asks ThirdParty if it wants to subscribe to LO2.
2. ThirdParty replies with `OK` and sends its subscription information back to SecondParty. The content of the subscription can be found here.

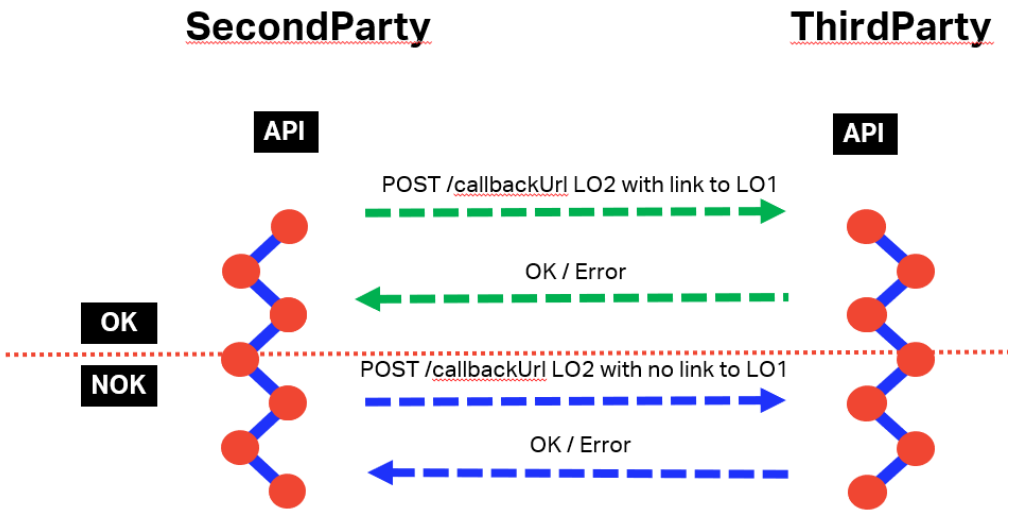


3. Delegation flow between SecondParty and FirstParty in order to give `GET` access to LO1 to ThirdParty. More information about the delegation can be found here.



Important: If FirstParty does not want to give access of LO1 to ThirdParty, SecondParty should not share the link to LO1 in LO2.

4. SecondParty sends notification containing the URI of the LO2 (containing link to LO1) on his ONE Record Server to the callback endpoint on the ThirdParty ONE Record Server.
5. ThirdParty effectuates a `GET` LO2 on the SecondParty ONE Record Server in order to retrieve the full content of the LO2.
6. ThirdParty effectuates a `GET` LO1 on the FirstParty ONE Record Server in order to retrieve the full content of the LO1.



Step 6 – FirstParty updates LO1

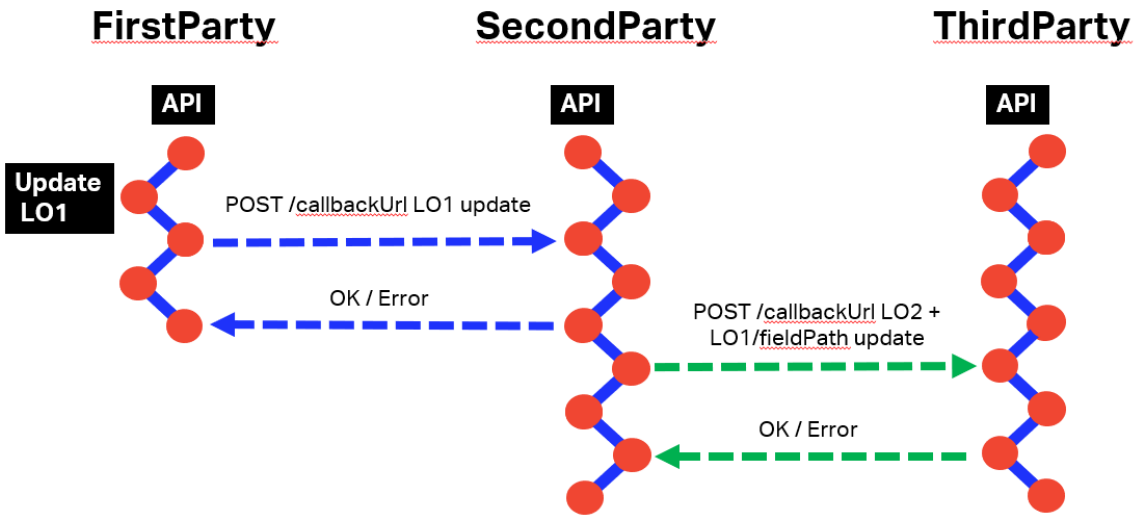
FirstParty updates LO1 on its own ONE Record Server, using `PATCH` specifications of the ONE Record standard.

Step 7 – FirstParty sends notification to SecondParty that the LO1 was updated

SecondParty sends `OBJECT_UPDATED` notification containing the URI of the LO1 on his ONE Record Server to the callback endpoint on the SecondParty ONE Record Server. The notification contains also the path of the field that has changed.

Step 8 – SecondParty may send notification to ThirdParty that the LO2 (which contains a link to LO1) was updated

SecondParty sends `OBJECT_UPDATED` notification containing the URI of the LO2 on his ONE Record Server to the callback endpoint on the ThirdParty ONE Record Server. The notification contains also the path of the field that has changed (in this case `LO1/fieldPath`).



There are use cases in which SecondParty doesn't notify ThirdParty when LO1 is changed and the ThirdParty doesn't get the updated information.

SecondParty decides by its own internal processes (for example an internal rule engine) when it is necessary to inform ThirdParty that LO1 was updated. Usually, this would happen when the LO1 update triggers an action,

e.g. object ready for pickup. If the LO1 update does not trigger an action for the ThirdParty, there is no need for notification between SecondParty and ThirdParty.

What if a Logistics Object contains multiple embedded links to other Logistics Objects?

The same mechanism applies in case of multiple embedded Logistics Objects.

In delegation scenario, what happens if FirstParty doesn't want to grant POST access to ThirdParty?

In delegation scenario, in case FirstParty does not want ThirdParty to `POST` updates, then the message from ThirdParty to SecondParty that something is wrong should be passed through traditional ways (e.g. email, phone, etc).

Data sovereignty is one of the base principles of ONE Record and the owner has the right to deny access to its data.

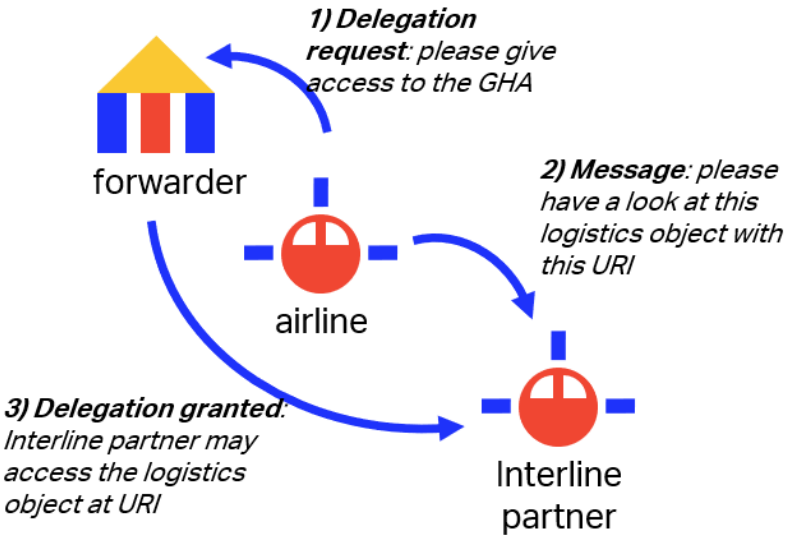
Conclusion

Publish & Subscribe only exists between the sharer and the immediate subscribers (in our case FirstParty - SecondParty) and third, fourth, n-th parties get delegated `GET` access, and not via Publish & Subcribe. Each party in between has the possibility of updating through their existing subscribers list the next party in the chain.

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
```

HTTP Headers

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

Header	Description
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
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

```
{
    "@context": {
        "@vocab": "http://onerecord.iata.org/"
    },
    "@type": "DelegationRequest",
    "targetLogisticsObject":"https://1recordserver.org/my_airline/airwaybill_123",
    "targetCompany": "https:/onerecordserver.org/delegated_company_identifier",
    "action": "DELEGATE", // or REVOKE
    "operations": "GET"
}
```

Where:

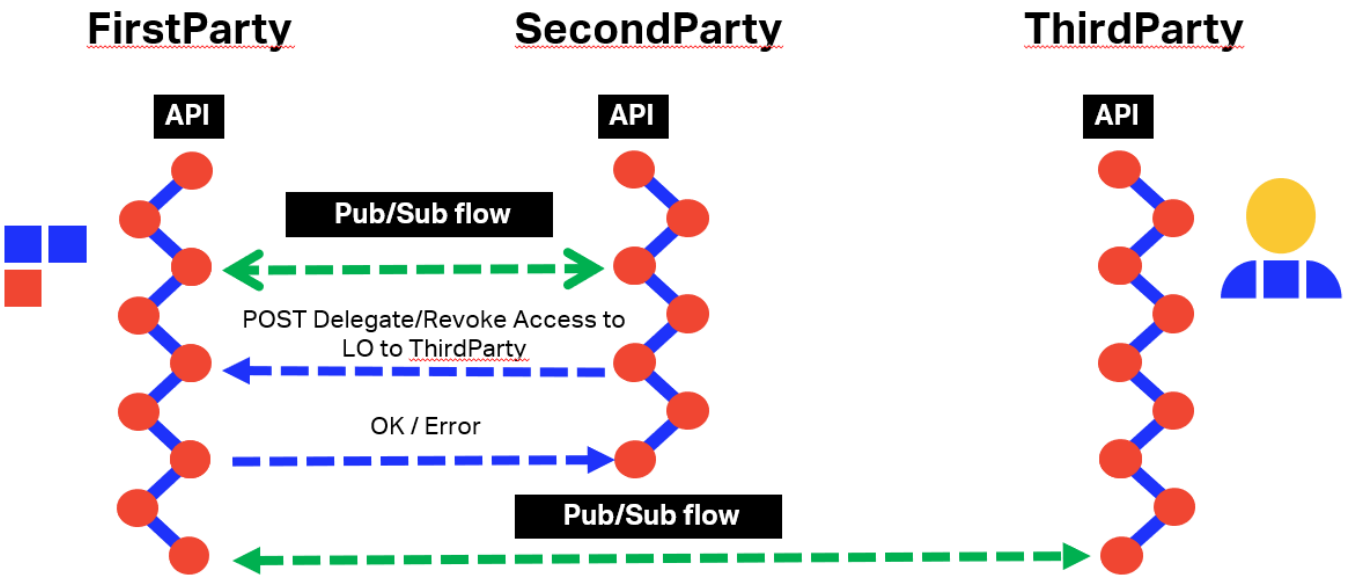
Field	Description	Required
@type	DelegationRequest	YES
targetLogisticsObject	The identifier of the logistics object to which the access is requested	YES
targetCompany	The party that receives the delegated rights	YES
action	The action to perform: `REVOKE` or `DELEGATE`	YES
operations	The API operations to which the access is requested: `GET`, `PATCH`, or both.	YES

Response

Code	Description	Response body
204	Request for delegation was successful	No response body
401	Not authenticated	Error model
403	Not authorized to send a Delegation request	Error model

In the following use cases, we consider three parties: **FirstParty**, **SecondParty** and **ThirdParty**.

Use Case 1



Delegation Scenario 1

Step 1 – Publish & Subscribe flow between FirstParty and SecondParty

First step contains a Publish & Subscribe flow between FirstParty and SecondParty for a Logistics Object LO with is created on the FirstParty ONE Record Server.

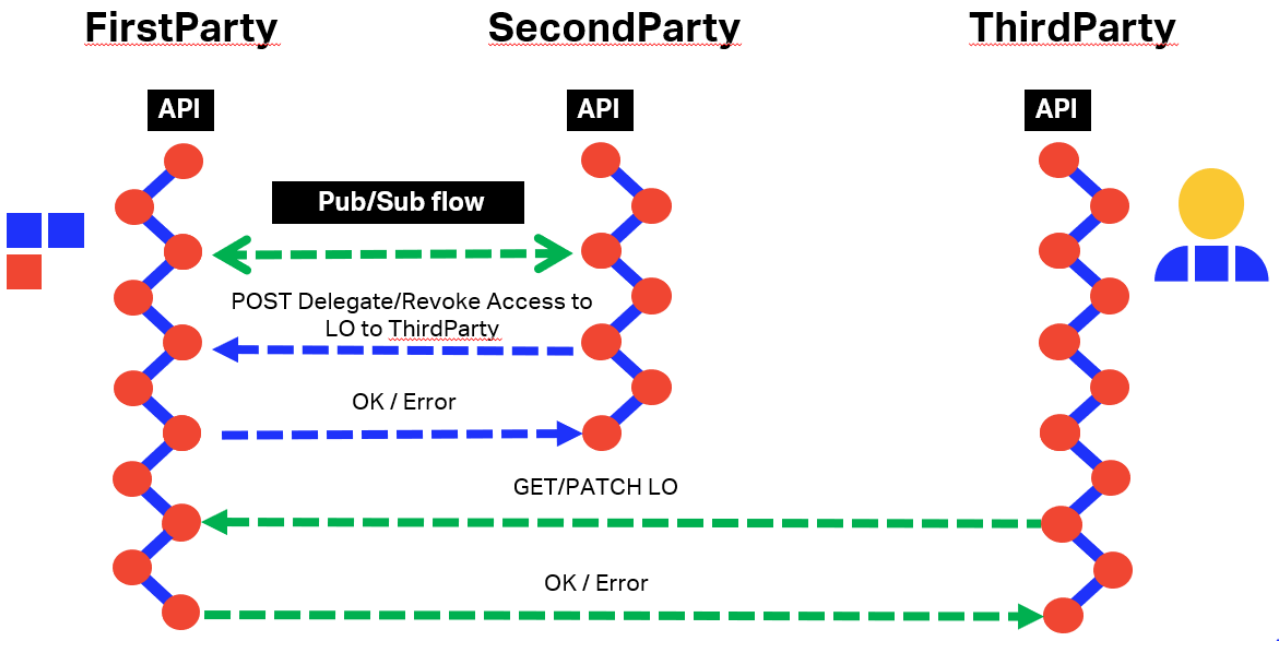
Step 2 – SecondParty requests delegation access to LO for ThirdParty

The SecondParty sends a delegation request to the FirstParty in order to grant ThirdParty access to LO (`GET`, `PATCH` or both).

Step 3 – Publish & subscribe flow between FirstParty and ThirdParty

If FirstParty decides to grant ThirdParty access to LO, then it initiates a Publish & Subscribe flow that would allow ThirdParty get notifications related to LO.

Use Case 2



Delegation Scenario 2

Step 1 – Publish & Subscribe flow between FirstParty and SecondParty

First step contains a Publish & Subscribe flow between FirstParty and SecondParty for a Logistics Object LO with is created on the FirstParty ONE Record Server.

Step 2 – SecondParty requests delegation access to LO for ThirdParty

The SecondParty sends a delegation request to the FirstParty in order to grant ThirdParty access to LO (`GET`, `PATCH` or both).

Same Request/Response as in Step 2 – SecondParty requests delegation access to LO for ThirdParty.

Step 3 – ThirdParty effectuates a GET or PATCH request on LO to FirstPArty

If FirstParty decides to grant ThirdParty access to LO, then ThirdParty can perform a `GET` or `PATCH` request on the LO to FirstParty.

Events (status update)

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

Create event (POST)

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

HTTP Headers

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

Header	Description
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
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

```
{
    "@context": {
        "@vocab": "http://onerecord.iata.org/"
    },
    "@type": "Event",
    "logisticsObjectRef": "https://1recordserver.org/my_airline/shipment_123456",
    "performedBy": "https:/1recordserver.org/some_company_identifier",
    "event": "DEP",
    "eventTimestamp": "2020-03-09T14:05:17+00:00",
    "eventApplicableTo": ["https://1recordserver.org/my_airline/booking_123456"],
    "location": {
      "@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"
        }
      }
    }
}
```

Where:

Field	Description	Required
@type	Event	YES
logisticsObjectRef	Logistics object the event is valid for	YES
performedBy	Company IoL identifier of the entity from which the event comes from	YES
event	Valid event	YES
eventTimestamp	Date and time when the event occurred	YES

eventApplicableTo	List of IRI, used only when the event is only applicable to certain linked logistics objects in the primary logistics object (logisticsObjectRef)	NO
location	Location 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 response body
400	Invalid Event	Error model
401	Not authenticated	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 (GET)

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

HTTP Headers

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

Header	Description
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

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	Events retrieved successfully	Event
401	Not authenticated	Error model
403	Not authorized to retrieve Events	Error model
404	Logistics Object Not Found	Error Model

Access Control

In ONE Record, access to resources could be handled by using Access Control Lists (ACLs) stored in the backend systems of the ONE Record Servers and defined using the Web Access Control standard from W3C.

Web Access Control

According to W3C specifications, Web Access Control it is a standard that enforces access control based on the Access Control List (ACL) RDF resource associated with the requested resource. It's concerned with giving access to agents (users, groups and more) to perform various kinds of operations (read, write, append, etc) on resources. In Web Access Control, an ACL consists of a set of **Authorizations**. Each Authorization is a single rule for access, such as "`entities one and two may write to resource someresource`", described with a set of RDF properties.

Access Control List Resources

In a system that uses Web Access Control, each web resource has a set of **Authorization** statements describing:

7. Who has access to that resource (that is, who the authorized **agents** are).
8. What types (or **modes**) of access they have.

These Authorizations are either explicitly set for an individual resource, or (more often) inherited from that resource's parent folder or container. In either case, the Authorization statements are placed into separate WAC documents called `Access Control List Resources` (or simply `ACLs`).

An Authorization is written as a series of RDF triples, with the Authorization resource as the subject for each of these triples. The Authorization resource URI is a hash URI, because of the requirement for potentially multiple, distinct Authorizations to be included in a single ACL resource and it has an `rdf: type` of `http://www.w3.org/ns/auth/acl#Authorization`.

The complete WAC ontology can be found here.

The location of the `acl` for a given resource may be discovered via a `Link` header with relation `rel="acl"`. Given a URL for an individual resource or logistics object, a client can discover the location of its corresponding ACL by performing a `GET` request and parsing the `rel="acl"` link relation.

Example:

GET `http://myServer/myAirline/logisticsObject` would return:

```
HTTP/1.1 200 OK
Link: <http://myServer/myAirline/logisticsObject/acl>; rel="acl"
```

If a resource does not have an individual ACL (and therefore relies on an implicit ACL from a parent), this link header will still be present, but will return a 404.

Clients MUST NOT assume that the location of an ACL resource can be deterministically derived from a document's URL.

Example of Authorizations

Single Authorization

Below is an example ACL resource that specifies that Party1 (as identified by its ONE Record Company Identifier `https://party1.server.com/company`) has full access (Read, Write and Control) to one of its web resources, located at `https://party1.server.com/company/logisticsObject`.

```
# Contents of https://party1.server.com/company/logisticsObject.acl
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://party1.server.com/company>;  # Company Identifier in the IoL
    acl:accessTo  <https://party1.server.com/company/logisticsObject>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```

Agent = Company Identifier in the Internet of Logistics

In this case, Party1 is the Owner of the Logistics Object. Owners are agents that have Read, Write and Control permissions.

Group Authorization

```
# Group authorization, giving Read/Write access to two groups, which are
# specified in the 'work-groups' document.
<#authorization2>
    a               acl:Authorization;
    acl:accessTo    <https://party1.server.com/company/logisticsObject2>;
    acl:mode        acl:Read,
                    acl:Write;
    acl:agentGroup  <https://party1.server.com/company/groups#Accounting>;
    acl:agentGroup  <https://party1.server.com/company/groups#Management>.
```

Corresponding `work-groups` Group Listing document:

```
# Contents of https://party1.server.com/company/groups
@prefix    acl:  <http://www.w3.org/ns/auth/acl#>.
@prefix     dc:  <http://purl.org/dc/elements/1.1/>.
@prefix  vcard:  <http://www.w3.org/2006/vcard/ns#>.
@prefix    xsd:  <http://www.w3.org/2001/XMLSchema#>.

<#Accounting>
    a               vcard:Group;
    vcard:hasUID    <urn:uuid:8831CBAD-1111-2222-8563-F0F4787E5398:ABGroup>;
    dc:created      "2018-09-11T07:18:19+0000"^^xsd:dateTime;
    dc:modified     "2019-08-08T14:45:15+0000"^^xsd:dateTime;

    # Accounting group members:
    vcard:hasMember  <https://party2.server.com/company5>;
    vcard:hasMember  <https://party3.server.com/company7>.

<#Management>
    a               vcard:Group;
    vcard:hasUID    <urn:uuid:8831CBAD-3333-4444-8563-F0F4787E5398:ABGroup>;
```

```
    # Management group members:
    vcard:hasMember  <https://party4.server.com/company1>.
```

Authenticated Agents (Default)

Authenticated access is a bit like public access, but it is not anonymous. Access is only given to clients who have logged on and provided a specific Company Identifier. This allows the server to track the entities who have used the resource.

```
@prefix   acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization2>
    a               acl:Authorization;
    acl:agentClass  acl:AuthenticatedAgent;
    acl:mode        acl:Read;
    acl:accessTo    <https://party1.server.com/company/logisticsObject>.
```

An application of this feature is to throw a resource open to all authenticated parties for a specific amount of time, accumulate the list of those who case as a group, and then later restrict access to that group, to prevent spam.

Modes of Access

The `acl:mode` predicate denotes a class of operations that the agents can perform on a resource.

`acl:Read`
This includes access to HTTP verb `GET`.

`acl:Write`
This includes `POST`, and `PATCH`. (`PUT` and `DELETE` are out of scope of ONE Record).

`acl:Control`
All methods, if the request URI is an ACL.

Out of scope of ONE Record

`acl:Append`
gives a more limited ability to write to a resource - Append-Only.

Inheritance

Use `acl:default`.

If you use `acl:accessTo` to protect a container, and add an `acl:default` predicate, that authorization rule by default will also apply to any of that container's children, unless that child has its own ACL.

The second is to use the `acl:accessToClass` property to state that the authorization rule applies to any resource with the named RDF type.

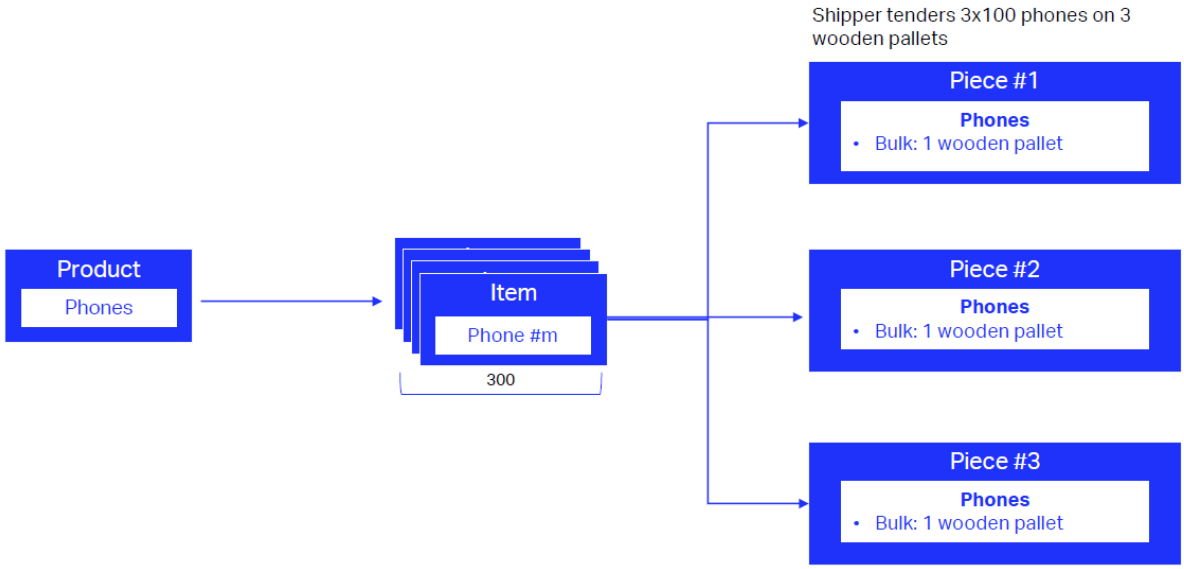
Delegation

When delegating access to a resource to a third party, a new Authorization element should be added to the ACL.

Use case

In this illustrative scenario, the entities would be **Shipper** (which has its own ONE Record Server) and **Forwarder**.

- Shipper creates a Shipment logistics object. Shipper is the owner of the phones, which constitute the Product. Each Product with a series number is an Item.
- Shipper packages the phones on wooden pallets, creates Pieces and handles them to Forwarder.



- Forwarder is responsible to create/update Transport Segment data for each Piece.

```
# Contents of https://shipperserver.com/company/shipment.acl
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://forwarderserver.com/company>;  # Forwarder Identifier in the IoL
    acl:accessTo  <https://shipperserver.com/company/shipment/transportSegment>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```

- Each Piece contains other information such as Goods Description which is only created/updated by Shipper.

```
# Contents of https://shipperserver.com/company/shipment.acl
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://shipperserver.com/company>;  # Shipper Identifier in the IoL
```

```
    acl:accessTo  <https://shipperserver.com/company/shipment/goodDescription>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```

- UPID information inside Piece could either be created by Shipper, in which case Shipper would be the owner of this data element. However, Shipper could handle the UPID creation/update to Forwarder, in which case Forwarder would be the owner of this data, meaning that only Forwarder could update this information.

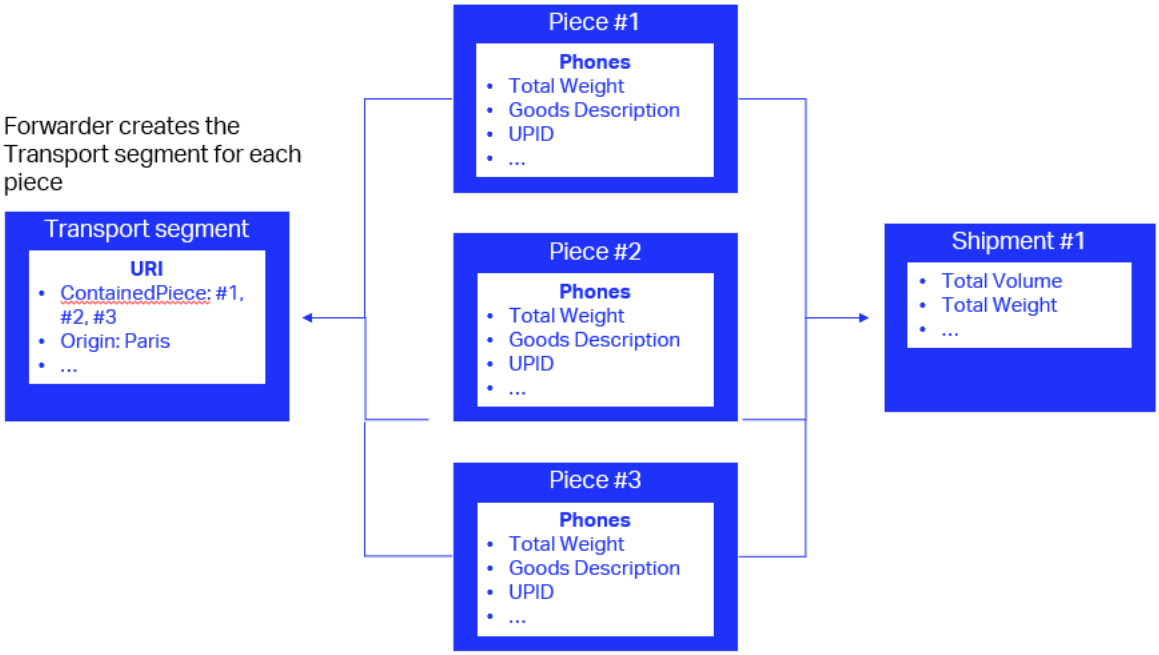
```
# Contents of https://shipperserver.com/company/shipment.acl
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://forwarder.com/company>;  # Forwarder Identifier in the IoL
    acl:accessTo  <https://shipperserver.com/company/shipment/UPID>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```

- Same principle as UPID could apply for Total Weight element inside Piece.

```
# Contents of https://shipperserver.com/company/shipment.acl
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<#authorization1>
    a             acl:Authorization;
    acl:agent     <https://forwarder.com/company>;  # Forwarder Identifier in the IoL
    acl:accessTo  <https://shipperserver.com/company/shipment/totalWeight>;
    acl:mode      acl:Read,
                  acl:Write,
                  acl:Control.
```



Create ACL (POST)

Request

HTTP Request type: **POST**

```
POST logisticsObjectURI/acl
Host: myonerecordserver.net
Authorization: mybearertokenbase64encoded
Accept: application/ld+json
```

Example:

POST https://www.onerecordcargo.org/my_airline/shipment_123456/acl

ONE Record does not define a specific model for ACL, but suggests the utilization of Access Control Ontology defined by W3C.

HTTP Headers

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

Header	Description
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
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

Response

Code	Description	Response body
201	ACL has been published for a Logistics Object	No response body
400	Invalid ACL	Error model
401	Not authenticated	Error model
403	Not authorized to publish an ACL	Error model
415	Unsupported Content Type	Error Model

Retrieve ACL (GET)

Request

HTTP Request type: **GET**

```
GET logisticsObjectURI/acl
Host: myonerecordserver.net
Authorization: mybearertokenbase64encoded
Accept: application/ld+json
```

Example:

GET https://www.onerecordcargo.org/my_airline/shipment_123456/acl

HTTP Headers

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

Header	Description
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

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

Code	Description	Response body
200	ACL returned successfully	ACL
401	Not authenticated	Error model
403	Not authorized to retrieve ACL	Error model
404	Logistics Object Not Found	Error Model

Bibliography:

Access Control Ontology: https://www.w3.org/ns/auth/acl

Web Access Control Specification from W3: https://www.w3.org/wiki/WebAccessControl

Web Access Control specifications by Solid project: https://github.com/solid/web-access-control-spec

Fedora project: https://wiki.lyrasis.org/display/FEDORA51/WebAC+Authorizations

Access Control in Linked Data Using WebID: https://arxiv.org/pdf/1611.03019.pdf

VCard ontology: https://www.w3.org/2006/vcard/ns#%3E

Context-Aware Access Control and Presentation of Linked Data: https://tel.archives-ouvertes.fr/tel-00934617/document

Security in ONE Record

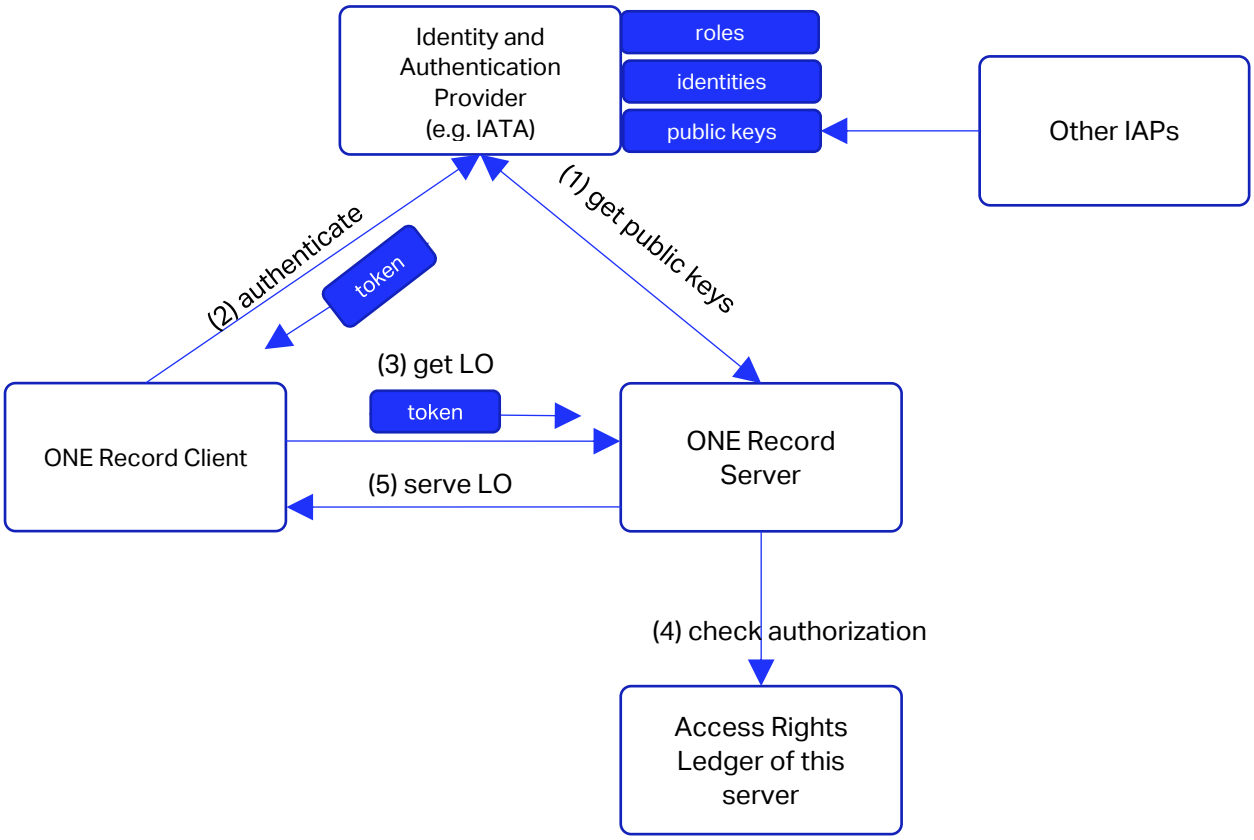
ONE Record as a basis for data sharing in the Internet of Logistics needs to ensure that data sharing is secure, i.e. that the participants sharing data are known, identified, authenticated and authorized for data access.

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.

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":
"0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L
6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ_2W-
5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-
65YGjQR0_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbISD08q
NLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqbw0Ls1jF44-csFCur-
kEgU8awapJzKnqDKgw",
          "e":"AQAB",
          "alg":"RS256",
          "use": "sig",
          "key_ops": "verify",
          "kid":"dcf1"
         },
         {
          "kty":"RSA",
          "n":
"0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L
6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ_2W-
5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-
65YGjQR0_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbISD08q
NLyrdkt-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://widgetco.net/lelmeeab", // The IOL identifier of the server the
subject belongs to
    "https://github.com.org/IATA-Cargo/ONE-Record/schema/role": "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:

- Oauth Token Binding
    - **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
    - **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
    - **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
    - **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. Promote standard libraries in Logistics community to mitigate implementation robustness risks.

Oauth Token Binding 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 requires all users to present client certificates which results in a lot of friction. The OAuth 2.0 Authorization Framework: JWT Pop Token Usage is similar to A Method for Signing HTTP Requests for Oauth, 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:
    1. Includes user and company user (or client) belongs to in token.
    2. Includes the clients public key identifier as registered in the IAP.
    3. Signed by the IAP.
2. Consumer requests an LDI/One Record service and includes in the request:
    1. PoP token which has the following characteristics:
        1. Includes the access token.
        2. Includes the resource that is being requested.
        3. Time and date of request.
        4. Signed by client with the matching private key of the public key registered for that client (and included in the access token).
3. LDI/One Record service upon receiving a request
    1. Validates the Access token.

2. 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.
3. 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.

Glossary

Term	Description
ACL	Access Control List
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
WAC	Web Access Control