

NSP API Developer Guide

- [Authentication](#)
 - [Accesstoken lifecycle](#)
- [HTTP Error Codes and Headers](#)
- [REST API with Access Token](#)
- [Notify Subscribers API before real-time streaming](#)
 - [NSP Streamable Events](#)
 - [Notify Subscriber API Payload](#)
 - [Posting Notify Subscriber payload with Filters](#)
 - [Getting Subscriber by SubscriberID](#)
 - [Getting Filter by FilterID](#)
 - [Updating a Notify Subscriber](#)
 - [Updating a Notify Filter](#)
 - [Deleting Notify Subscribers and Notify Filters](#)
- [Real Time Streaming API](#)
 - [Endpoint](#)
- [Advanced Filtering](#)
 - [JSON Path Expressions](#)
 - [JSON Path & Filter Expression examples for VesselVisit payload](#)
 - [Difference Between JSON Path Expressions and JSON Filter Expressions](#)
 - [Json Paths](#)
 - [Json Filter Expressions](#)
 - [Json Filter Expressions With Logical AND/OR](#)
 - [Conditional Filtering on Payload Content](#)
 - [Event Name Filtering](#)
 - [Simple Predicate Filtering](#)
 - [Using Change Fields Filtering](#)
 - [Combining changeField and predicate Filtering](#)
 - [Filtering with Advanced Predicates via JSON Path Filter Expressions](#)
 - [Combining changeFields Filter and Advanced Predicates](#)
 - [Modifying Payload Content](#)
- [Child Entity Use Case](#)
- [Instance Security in Rest API](#)
 - [Predicate Name](#)
 - [Predicate Value](#)
 - [Example](#)
 - [Rest API](#)
 - [Business Units by Role Rest Endpoint](#)
 - [Request](#)
 - [Endpoint](#)
 - [Query Parameters](#)
 - [Example](#)
 - [Response](#)
 - [Example](#)

Authentication

NSP implemented OAuth2 with JWT as a access token, application need to call OAuth2 authentication endpoint and retrieves the access_token part of the json. **client_id** and **client_secret** will be distributed separately to access NSP API's.

Each application will have certain scopes so that when application access different REST endpoint, scope will be enforced

Authentication Endpoint
<pre>curl -v --location -X POST \ "https://SERVERNAME/auth/realms/nsp/protocol/openid-connect/token" --header "Content-Type: application/x-www-form-urlencoded" \ --data-urlencode "client_id=<clientId will be distributed separately>" \ --data-urlencode "client_secret=<client secret will be distributed separately>" \ --data-urlencode "grant_type=client_credentials"</pre>
Authentication Endpoint Response
<pre>{"access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXZWl1a2lkIiA6ICJwLh5bTzoQutDa1Y5VUE4dVhkX0JWanpUczdYLRienpVcc" "expires_in":1800, "refresh_expires_in":43200, "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWl1a2lkIiA6ICJhNDC4ZTI1MC0wYWlyLTQ4YjktYTE3OC1kZTRjNzEzZDVhYj"}</pre>

```
"token_type": "bearer",
"not-before-policy": 0,
"session_state": "3b4de221-cda2-4c46-9b62-79deefbf2bdc",
"scope": "nsp.YILPORT_GEMLIK.all.all"}
```

Access token lifecycle

OAuth2 JWT access token has certain lifecycle, most of the programming languages have a libraries to handle it. Very high level

- Application needs to check expires_in (in the above example it expires in 30min) and using refresh_token need to refresh access token.
- In order to refresh access token, application needs to call following URL with **grant_type=refresh_token**

Refresh token Endpoint

```
curl -v --location -X POST "https://SERVERNAME/auth/realms/nsp/protocol/openid-connect/token" \
--header "Content-Type: application/x-www-form-urlencoded" \
--data-urlencode "client_id=<clientId will be distributed>" \
--data-urlencode "client_secret=<client secret be distributed>" \
--data-urlencode "refresh_token=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNDC4ZTI1MC0wYWYyLTQ4YjktYT
```

- Above endpoint produces same payload as authentication endpoint
- Refresh token expires in every 12hrs (refresh_expires_in field in the above example) and which require a new call to the authentication endpoint with **grant_type=client_credentials**

HTTP Error Codes and Headers

40X - If there is authentication failure, client will receive 40x error code. It could be either 400 or 401 or 403

200 - http code for all the successful execution

50X - http code appears when server not able to process the request

REST API with Access Token

All the rest api's require a authentication header with valid access token to receive the data. Application responsibility to append the Authentication header, if it fails it will receive one of the auth error codes.

Sample REST api call with authentication header

Sample rest api with auth token

```
curl -X POST "https://SERVERNAME/billing/unitevents?operator=<valid operator>" -H "accept: */*" -H "Content-Type:
-H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJwWlh5bTZoQutDa1Y5VUE4dVhkX0JWanpUczdY
```

Notify Subscribers API before real-time streaming

Before using realtime streaming over websocket, the application needs to subscribe to events. These subscribed events will be pushed over a websocket connection. If there is no subscription, no events will be streamed to the client. Look for detailed description and list of Notify Subscribers API in Swagger UI.

NSP Streamable Events

The following list of events is the possible N4-NSP events that can be subscribed to and have a websocket stream:

Agent
AppointmentQuotaRule
AppointmentTimeSlot
BillOfLading
BIItem
Booking
CarrierItinerary
CarrierService
ChargeableAppointmentEventPayload
ChargeableMarineEvent
ChargeableUnitEvent
Che
ChePool
Complex
DynamicEntity
EcEvent
EdiBatch
EdiTradingPartner
EqBaseOrder
EqBaseOrderItem
EquipmentOrder
EquipmentOrderItem
EquipType

Event
EventType
Facility
Flag
FlagType
GateAppointment
GenericEntity
HandlingCode
HazardItem
Hazards
ItemServiceType
ItemServiceTypeUnit
LineOperator
MoveEvent
Operator
PackageType
Placard
PointCall
PointOfWork
ProductType
RailcarPlatform
RailcarVisit
RailRoad
RefCountry
RefState
ServiceOrder
ServiceOrderItem
Shipper
TbdUnit
Topology
TrainVisit
TruckingCompany
TruckTransaction
TruckVisit
TruckVisitAppointment
TruckVisitStats
Unit
UnitRecap
User
Vessel
VesselBerthing
VesselClass
VesselVisit
VesselVisitLine
Veto
VisitItinerary
WorkAssignment
WorkInstruction
WorkQueue
WorkShift
Yard

The following list are comprised of streamable N4 analytics events, these can be subscribed to but predicates and field filtering is not allowed.

CarrierProductivity
ConsumerState
CranesIntensity
CraneProductivityStatistics
UnitFullMyRecap
UnitRecapFull
UnitRecapFullContent
LastPow

Notify Subscriber API Payload

To POST a new notify subscriber, a well-structured JSON is needed:

Accepted JSON with all the fields:

Request Body for POST Notify Subscriber

```
{
  "clientId": "",
  "type": "",
  "channel": "",
  "groupId": "",
  "persistence": "",
  "transport": "",
  "extras": {
    "additionalProp1": {}
  },
  "filters": [
    {
```

```

        "eventName": "",
        "filter": {
            "predicate": [],
            "fields": ""
        },
        "operation": "",
        "changeFields": {
            "include": [],
            "exclude": []
        }
    }
},
"scope": {
    "yard_gkey": "",
    "yard_id": "",
    "facility_gkey": "",
    "facility_id": "",
    "complex_gkey": "",
    "complex_id": "",
    "operator_gkey": "",
    "operator_id": "",
    "instance_id": "",
    "level": ""
}
}

```

clientId: Id of the client (It appears part of the security token).

type: subscribe or produce (required)

channel: Described channel to communicate the NSP application with. By default it uses **"/nsp"** (required)

groupId: Id of the group (required)

persistence: A boolean value that determines if a subscription should be deleted or not when the websocket connection is closed (optional)

transport: Type of transport for subscribed events: ws (websocket), webhook (required)

extras: Additional information which may possibly be needed based on transport type. Websocket transport type, we don't need extras. (optional)

filters: a list of filter JSON objects (required)

Filter object:

1. **eventName:** the business event to filter over, example: CharableUnitEvent, VesselVisit or ChargeableMarineEvent (required field)
2. **operation:** operation type, create, update, delete (optional)
3. **changeFields:** A map of two keys, include and exclude which take a list of fields from the event payload (optional)
4. **filter:** a JSON mapping of predicate and fields desired for this Notify Filter (optional)
 - a. **predicate:** a comma-separated list of Strings with a JSON path expression and the value it should match to. Example for a Unit predicate: "predicate": ["visitState=1ACTIVE", "category=STRGE"] (optional)
 - b. **fields:** a String containing comma-separated JSON path expressions to indicate which fields should be returned Example for a Unit field list (optional)

scope: part of the scope we accepts instance_id, operator_id etc., (optional - this is retrieved from the access token, clients don't need to send these parameters)

Refer to the sample request and response body below when posting a subscription (with or without filters)

Examples:

Posting Notify Subscriber payload with Filters

Posting a notify subscriber that contains two filters, one for ChargeableUnitEvents and one for ChargeableMarineEvent.

This would be the request body, with two separate filters in the "filters" list.

Request Body

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "ChargeableUnitEvent"
    },
    {
      "eventName": "ChargeableMarineEvent"
    }
  ]
}

```

Create Notify Subscriber

```
curl -X POST "https://SERVERNAME/notify/subscribers" -H "accept: */*" -H "Content-Type: application/json" \
-H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOjAiAislDUiiwia2lkIiA6ICJwLlh5bTZOQTda1Y5VUE4dVhkX0JWanpUczdYL" \
-d "{\"type\":\"subscribe\",\"channel\":\"/nsp\", \"groupId\":\"sampleGroupId\", \"transport\":\"ws\", \"filters\":{\"
```

Response Body

```
{
  "id": "sub-4d5tfspjgvsk6",
  "clientId": "nsp-client",
  "type": "subscribe",
  "groupId": "sampleGroupId",
  "persistence": null,
  "channel": "/nsp",
  "transport": "ws",
  "extras": null,
  "filters": [
    {
      "filterId": "ftr-b9dc1cncivts7",
      "eventName": "ChargeableUnitEvent",
      "filter": null,
      "operation": null,
      "changeFields": null
    },
    {
      "filterId": "ftr-a9lttuvkmvsrf",
      "eventName": "ChargeableMarineEvent",
      "filter": null,
      "operation": null,
      "changeFields": null
    }
  ],
  "scope": {
    "yard_gkey": null,
    "yard_id": null,
    "facility_gkey": null,
    "facility_id": null,
    "complex_gkey": null,
    "complex_id": null,
    "operator_gkey": null,
    "operator_id": null,
    "instance_id": "DPW_RWG",
    "level": "GLOBAL"
  }
}
```

Example with predicates for ChargeableUnitEvents, looking for events that are billable and the freight kind is "FCL".

Request Body for ChargeableUnitEvent with Predicates

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "ChargeableUnitEvent"
      "filter": {
        "predicate": ["isBillable=true", "freightKind=FCL"]
      }
    }
  ]
}
```

```
    }
  }
]
```

Getting Subscriber by SubscriberID

To request a specific notify subscriber, the path variable subscriberId will be needed.

Here the subscriberId is sub-7vkf1d4mgbtuv.

Get Notify Subscriber by subscriberId

```
curl -X GET "https://SERVERNAME/notify/subscribers/sub-7vkf1d4mgbtuv" -H "accept: */*" -H "Authorization: Bearer e
```

The response body below contains information about the subscriber and the filters it contains.

Response for subscribers by subscriberId

```
{
  "id": "sub-7vkf1d4mgbtuv",
  "clientId": "client1",
  "type": "subscribe",
  "channel": "/nsp",
  "groupId": "sampleGroupId",
  "persistence": null,
  "transport": "ws",
  "extras": null,
  "filters": [
    {
      "filterId": "ftr-bcr9at25ffvm8",
      "eventName": "ChargeableUnitEvent",
      "filter": {
        "predicate": [
          "isBillable=true",
          "freightKind=FCL"
        ]
      },
      "operation": null,
      "changeFields": {}
    }
  ],
  "scope": {
    "yard_gkey": null,
    "yard_id": null,
    "facility_gkey": null,
    "facility_id": null,
    "complex_gkey": null,
    "complex_id": null,
    "operator_gkey": null,
    "operator_id": null,
    "instance_id": "instance1",
    "level": "GLOBAL"
  }
}
```

Getting Filter by FilterID

To request a specific filter, the path variables subscriberId and filterId will be needed.

Here the subscriberId is sub-7vkf1d4mgbtuv and the filterId is ftr-bcr9at25ffvm8

Get filters by subscriberId and filterId

```
curl -X GET "https://SERVERNAME/notify/subscribers/sub-7vkf1d4mgbtuv/ftr-bcr9at25ffvm8" -H "accept: */*" -H "Autho
```

This command has the response body of:

Response for subscribers by subscriberId and filterId

```
{
```

```

    "filterId": "ftr-bcr9at25ffvm8",
    "eventName": "ChargeableUnitEvent",
    "filter": {
      "predicate": [
        "isBillable=true",
        "freightKind=FCL"
      ]
    },
    "operation": null,
    "changeFields": {}
  }
}

```

Updating a Notify Subscriber

To update a notify subscriber, provide the path variable subscriberId and the new subscriber request body in a PUT request.

In this example we will be adding a new filter into this subscriber. This filter will filter over Unit events.

Update Notify Subscriber

```
curl -X PUT "https://SERVERNAME/notify/subscribers/sub-7vkf1d4mgbtuv" -H "accept: */*" -H "Content-Type: applicat
```

The request body on the data flag below:

Response for update subscribers by subscriberId

```

{
  "clientId": "client1",
  "type": "subscribe",
  "channel": "/nsp",
  "groupId": "sampleGroupId",
  "persistence": null,
  "transport": "ws",
  "extras": null,
  "filters": [
    {
      "filterId": "ftr-bcr9at25ffvm8",
      "eventName": "ChargeableUnitEvent",
      "filter": {
        "predicate": [
          "isBillable=true",
          "freightKind=FCL"
        ]
      }
    },
    {
      "eventName": "Unit"
    }
  ],
  "scope": {
    "instance_id": "instance1"
  }
}

```

Note: if the specific filterId for the existing filter is not provided that filter will be overridden even if the other fields remain the same.

The response body for this request fully describes the new notify subscriber and its filters.

Updating a Notify Filter

To update a notify filter, provide the path variables subscriberId and filterId and the new filter request body in a PUT request.

In this example, we change our filter on ChargeableUnitEvent with the predicates "isBillable=true" and "freightKind=FCL" to have "isBillable=false"

Update Notify Subscriber filter information

```
curl -X PUT "https://SERVERNAME/notify/subscribers/sub-7vkf1d4mgbtuv/ftr-bcr9at25ffvm8" -H "accept: */*" -H "Cont
```

The request body payload in the above -d flag:

Response for update subscribers by subscriberId and filterId

```
{
```

```

    "eventName": "ChargeableUnitEvent",
    "filter": {
      "predicate": [
        "isBillable=false",
        "freightKind=FCL"
      ]
    }
  }
}

```

The response body fully describes the edited filter.

Deleting Notify Subscribers and Notify Filters

To delete a notify subscriber provide the subscriberId as a path variable in a curl delete request.

Deleting notify subscriber with subscriberId as path variable


```
curl -X DELETE "https://SERVERNAME/notify/subscribers/sub-fjm659fcbrt8g" -H "accept: */*" -H "Authorization: Bearer "
```

To delete a specific filter within a notify subscriber, provide the subscriberId and filterId as path variables in a delete request.

Deleting filter in a notify subscriber with subscriberId and filterId as path variables

```
curl -X DELETE "http://SERVERNAME/notify/subscribers/sub-7vkf1d4mgbtuv/ftr-b45d411achuio" -H "accept: */*" -H "Aut
```

Real Time Streaming API

 NSP team developed a Java client library and sample spring boot application, which has details explanation how to use the library with Java code. Please read README.md file on these projects and start using it.

<https://git.navis-dev.com/projects/N4FRA/repos/nsp-client/browse>

<https://git.navis-dev.com/projects/N4FRA/repos/nsp-client-springboot/browse>

NSP provides real-time streaming over websocket. Before the application uses this feature, it needs to subscribe for events. These subscribed events will be pushed over websocket. If there is no subscription, no events will be streamed to the client.

Endpoint

clients can be connect to websocket(WS) at **wss://SERVERNAME/notify/stream?groupId=<required>&id=<required>&offset=<optional>**

groupId (required) - If clients need to retrieve the history where it left off, need to send the groupId part of the ws connection URL. Otherwise, backend will generate an internal group id which is unique for each WS connection

id (required) - In order to establish a websocket connection, requires a valid subscription id which was created in previous steps.

offset (optional) - If client expects a message from a certain offset, which needs to have **same groupId** part of the previous connection and valid offset sent via payload on the WS connection.

Before accessing WS endpoint, clients need to be authenticated and add an access token via http **Authorization** header

Sample WS call with auth token without groupId and offset

```
curl -v --include --no-buffer --header "Connection: Upgrade" --header "Upgrade: websocket" --header "Sec-WebSocket-
https://SERVERNAME/notify/stream?groupId=validgroupidwithstrings&id=sub-7vkf1d4mgbtuv
```

Note: 1. After establishing websocket connection, client needs to send periodic ping websocket messages, otherwise load balancer will disconnect idle connections (idle timeout 10min)

2. clientId needs to be matched from subscriptions API's and streaming endpoint. Which means client id used to generate jwt token needs to be matched for security reasons.

Payload structure which receives on WS will be different from REST. WS payload has more data compare to the REST payload

Sample ws payload

```

{"businessEntity":"ChargeableUnitEvent",
 "operation":"update",
 "changes":[
   {"field":"isoCode", "old":"","new":"2200"}
 ],
 "payload":{
   "keys":[
     {"entityName":"ChargeableUnitEvent", "id":17, "gkey":30}
   ]
 }
}

```

```

    },
    "content":{"content of above business entity"}
  },
  "extras" :{
    "filterId":"which filter qualified",
    "subscriptionId" : "which subscription fired this event",
    "offset":"offset of the message as long in number"
  }
}

```

businessEntity - Name of the business entity which subscribed for. Ex: ChargeableUnitEvent or VesselVisit etc.,

operation - What kind of operation create | update |delete

changes - Array of fields changed in the entity. It comes with old and new values for each field

payload - Which contains two parts keys and content

keys - Array of keys. Mostly gkey of given entity but few entities will receive more than one key

content - Content of the given entity

extras - Which contain extra information like which subscription triggered this event

subscriptionId - id of the subscription

filterId - id of the filter from above subscription

offset - offset of the message

Advanced Filtering

This section will describe in detail how to use and take advantage of the Notify Subscriber's filtering capabilities, with an emphasis on utilizing JSON Path Expression to request more filtered data.

Each Notify Subscriber can contain one or more Notify Filter. The Notify Filter contains simple filtering criteria such as **eventName** and **operation**. Both of these are single values, for **eventName** the user can provide a business entity event name such as "ChargeableUnitEvent" or "VesselVisit", the filter attached to this subscription will then only send out events related to the specific event. For **operation** the valid values are create, update, delete.

More involved filtering is available with the **changeFields** and filter object which contains the **predicates** and **fields**.

JSON Path Expressions

To fully utilize predicates and fields filtering, the Notify Filter's **predicate** and **fields** can be supplied with JSON path expressions and JSON filter expressions. The JSON path expressions referred to in this document are referring to the [Jayway JsonPath implementation](#) which is based on the [Stefan Goessner JsonPath implementation](#). This section will use a sample VesselVisit event and provide JSON Path expression examples to show how top-level elements, child elements, and JsonArrays can be requested. JSON path expressions with logical AND (&&) and OR (||) are also possible and will be explained.

The below VesselVisit event JSON was created by the Navis Smart Platform's API component when this Vessel "DOB196" had its estimated time of arrival (publishedEta) updated.

This JSON payload broken down:

- **businessEntity**: This element is the entity type that has a change. This is the field that is checked against the Notify Filter's eventName value.
- **operation**: This element is the type of change that occurred. This is the field that is checked against the Notify Filter's operation value.
- **changes**: This element is a JSON array consisting of what changes occurred. The field represents the entity's field that was changed and the new and old values are supplied as well. This element is checked against the Notify Filter's changeField object.
- **payload**: The payload consists of two elements
 - **keys**: This element consists of metadata about this object.
 - **content**: This element is representative of the entity's current state after the changes described have taken place. This content element is the JSON object which a Notify Filter's predicate and fields filtering may be written against.

VesselVisit Event

› [Expand source](#)

JSON Path & Filter Expression examples for VesselVisit payload

The Notify Filter's filter for predicate and fields will only be applied over the payload content of the overall JSON message. Because of this, JSON paths and filter expressions can be written with the payload.content as the top level JSON Object. In other words, there is no need to supply the strings "payload" or "content" into your JSON path or expressions. Those strings will automatically be applied.

Difference Between JSON Path Expressions and JSON Filter Expressions

JSON path expressions and JSON filter expressions are very similar, they both use the same logic and form when traversing a JSON document. However, a JSON path expression is a determinate expression, it can only return the specified value(s) the path expression finds in the document. If the path expression leads to nowhere, meaning the desired field is non-existent in the JSON then nothing is returned. A JSON filter expression on the other hand returns a boolean, true or false, this returned boolean is used to determine whether the JSON object at the level the filter expression is being evaluated will be returned.

Json Paths

To access top-level elements on the VesselVisit payload content, only the field name is needed.

Example: Want the path for the vessel class id (vesclassId) of our VesselVisit.

vesclassId

Returns: "DOB196"

*The full JSON path for vessel class Id (vesclassId) would be the below path, however for the Notify Filter we need \$.payload.content.vesclassId

Example: Want the estimated move counts (estMoveCount) for this VesselVisit.

estMoveCount

Returns:

```
"estMoveCount": {
  "estBbkDischarge": 23,
  "estBbkLoad": 13,
  "estDischarge": 11,
  "estLoad": 12,
  "estRestow": 14,
  "estShift": 10
}
```

*The estimated move counts (estMoveCount) element in our VesselVisit payload is its own object, but it is a direct

Example: Want the estimated restows (estRestow) associated for this VesselVisit.

estMoveCount.estRestow

Returns: 14

*The estimated restow count (estRestow) is an element within the child object estMoveCount, so the full path to the

Example: Want the path for all the lines of our VesselVisit.

Two answers:

lines

lines[*]

Returns:

```
[
  {
    "lineOutVoyNbr" : "52",
    "lineInVoyNbr" : "51",
    "lineId" : "MAT",
    "cargoCutoff" : null,
    "emptyPickup" : null,
    "beginReceive" : null,
    "reeferCutoff" : null,
    "hazCutoff" : null
  },
  {
    "lineOutVoyNbr" : "50",
    "lineInVoyNbr" : "55",
    "lineId" : "MAT",
    "cargoCutoff" : null,
    "emptyPickup" : null,
    "beginReceive" : null,
    "reeferCutoff" : null,
    "hazCutoff" : null
  }
]
```

*The path "lines" is sufficient to obtain lines associated with this vessel. Because the lines element is a JSON Ar

Example: Want the line outbound voyage numbers (lineOutVoyNbr) associated with the lines of our VesselVisit.

```
lines[*].lineOutVoyNbr
```

Returns:

```
[
  "52",
  "50"
]
```

The `"lines[]"` path is combined with the desired field. `"lines.lineOutVoyNbr"` is incorrect because the lines element

Json Filter Expressions

Json Path filter expressions are used to filter the JSON itself. The expressions are boolean and can only return true or false. The result of the expression determines whether the JSON in its entirety is returned or not.

The JSON expressions must be written in this format:

```
[?(<expression> )]
```

where "<expression>" should be replaced with the actual Json filter expression.

The filter expressions can evaluate for the following operations:

Operator	Description
==	left is equal to right (note that 1 is not equal to '1')
!=	left is not equal to right
<	left is less than right
<=	left is less or equal to right
>	left is greater than right
>=	left is greater than or equal to right
==~	left matches regular expression [?(@.name =~ /foo.*?/i)]
in	left exists in right [?(@.size in ['S', 'M'])]
nin	left does not exists in right
subsetof	left is a subset of right [?(@.sizes subsetof ['S', 'M', 'L'])]
anyof	left has an intersection with right [?(@.sizes anyof ['M', 'L'])]
noneof	left has no intersection with right [?(@.sizes noneof ['M', 'L'])]
size	size of left (array or string) should match right
empty	left (array or string) should be empty

To form these filter expressions, the `"@.<childValue>"` notation must be used.

`@` represents the current element being processed

`.<childValue>` represents the child element of the current element being processed

The following examples will re-use the VesselVisit payload from above:

Example: Want to only capture VesselVisits for Vessel DOB196, so where the vessel name (vesName) is DOB196

```
[?(@.vesName=="DOB196")]
```

Explanation:

- 1) The full filter expression is actually `$.payload.content.[?(@.vesName=="DOB196")]`
However as explained previously for Notify Filters, the payload's content JSON element is the only element available. The `"$.payload.content"` should be omitted.
- 2) The `@` represents the current element being processed, in **this case** it is the main level `"content"` element from
- 3) `"=="` is the equality operator, to match vessel name to the target Vessel name `"DOB196"`
- 4) `"DOB196"` is in **double** quotations, because the vesName field contains a String value. String values must be matched exactly.

Example: Want to only capture VesselVisits for Vessels that have more than 10 estimated restows (estRestow).

```
estMoveCount.[?(@.estRestow>10)]
```

Explanation:

- 1) Again, the full expression should be: `$.payload.content.estMoveCount.[?(@.estRestow>10)]`

Because the filter expression will be run against the payload content object, it can be omitted.
2) Estimated restow count is a child field of Estimated Move Count (estMoveCount), so the path to estRestow. Remember the "@" notation represents the current element being processed and estRestowCount is not a direct field

Example: Want to only capture VesselVisits for a Vessel that has a line with a line outbound voyage number (lineOutVoyNbr) equal to 52.

```
lines[?( @.lineOutVoyNbr == "52")]
lines[*].[( @.lineOutVoyNbr == "52")]
```

Explanation:

1) Both answers are correct and will **return** the same thing, because the lines element in the payload is an array of

Json Filter Expressions With Logical AND/OR

The JSON filter expressions above can be combined with a logical AND (&&) or OR (||). These logical conditions behave normally, an AND condition requires all expressions being evaluated to be true, an OR condition requires just one expression to be true.

An expression with a && or || condition is still only one expression, so they still conform to the syntax of:

[?(<expression>)]

Example: Want to capture VesselVisits for vessels that have a visit phase (visitPhase) of "40WORKING" or "60DEPARTED".

```
[?(@.visitPhase=="60DEPARTED" || @.visitPhase=="40WORKING")]
```

Explanation:

1) The expression is everything in between the parentheses. Either side of the OR (||) condition need to **return true**

Example: Want to capture VesselVisits that have the vessel in a visit phase (visitPhase) of "60DEPARTED" and the vessel line id (lineId) is "MAT"

```
[?(@.visitPhase=="60DEPARTED" && @.lineId=="MAT")]
```

Explanation:

1) Both sides of the AND (&&) condition need to **return true for this** VesselVisit to pass the expression.

Example: Want to capture VesselVisit that has a vessel with a line, who's line outbound voyage number (lineOutVoyNbr) is 52 and the line inbound voyage number (lineInVoyNbr) is 51.

```
lines[?(@.lineOutVoyNbr=="52" && @.lineInVoyNbr=="51")]
lines[*].[( @.lineOutVoyNbr=="52" && @.lineInVoyNbr=="51")]
```

Explanation:

1) Both sides of the AND (&&) condition need to **return true for this** VesselVisit to pass the expression.
2) Both syntax's are correct as lines is an array of line objects.

Conditional Filtering on Payload Content

Conditional filtering is based on the filter's changed fields, predicates, operation, and event name. If all present conditions are met and passed, then the filter will allow for this event's message to pass through. If a condition is present and does not pass, then the message will be filtered out and not be sent to the desired consumer. If even one condition fails, then the whole message is filtered out.

This section will go over the possible conditional filters that can be added and combined to customize a filter.

The following examples will show what is needed to POST a new subscriber. Each additional example will add more conditional filters. The example subscriptions and filters will be targeting a VesselVisit payload.

Event Name Filtering

This is the simplest filter, it is based on the filter's event name (eventName). This filter allows for the user to select the event type that happened in N4. For example, VesselVisit or ChargeableUnitEvent are event types that can be used to filter.

There can only be one event type entered per filter.

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
```

```
        "eventName": "VesselVisit"
      }
    ]
  }
}
```

Important part to focus on here is the filters array. Each curly bracket {} in the array can represent another Notify Filter. In this example there is only one Notify Filter. This Notify Filter only has one condition, which is the "eventName" must be "VesselVisit".

This filter will only send "VesselVisit" events and nothing else. All "VesselVisit" events will be sent because there are no other conditional filters present.

Simple Predicate Filtering

Predicate filtering is an assertion based filtering. The predicates provided must all be true in order for the "predicates" condition in the filter to pass.

There are two types of predicates that can be provided to the Notify Filters, the first is a simple key-value pair, where the key or field in the JSON payload must be equal to the value provided.

The structure for the simple predicate: "field=value", will find the field specified in the JSON value and its value in the JSON must equal the value supplied in the key-value simple predicate.

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit"
      "filter": {
        "predicate":["visitPhase=60DEPARTED", "vesselType=CELL", "lines[*].lineOutVoyNbr=52"]
      }
    }
  ]
}
```

Explanation:

This subscription now has a single filter, made up of an eventName and predicate filter. Note that the predicate filter exists within a "filter" object.

The predicate filtering exists as an array of strings. Each predicate needs to be enclosed within double quotations; additional quotations for string values are unnecessary.

The predicates on this filter are: the visit phase (visitPhase) must be 60DEPARTED, the vessel type (vesselType) must be CELL, and one of its' lines must have an outbound voyage number (lineOutVoyNbr) of 52.

The reason the lines predicate is structured as "lines[*].lineOutVoyNbr" and not "lines.lineOutVoyNbr" is because "lines" on the VesselVisit payload is an array of line objects. Lines itself does not contain a "lineOutVoyNbr" field, so we use the lines[*]

to specify for the line objects in the array.

Using Change Fields Filtering

Change Fields (changeFields) filtering is a filter based on the business entity's changed fields. When an entity is changed the field changed, the old, and the new value of the field are recorded in the business entity's payload. This changeFields portion of our filter allows for filtering over which fields have been changed. The changeFields filter consists of an included and excluded list of fields to filter over, each list should contain either field names or asterisk * indicating all fields.

The included changeFields list takes precedence, if any field that has been changed for a business entity is present in both the included list and excluded list, this business entity will not be filtered out and will be sent. If the current event has changed fields that appear only in the excluded changeFields' lists, the event will not be sent. If the included list contains just the asterisk (indicating all fields) and there is at least one field from the changed fields that doesn't appear in the excluded changeFields's list then the event will be sent. If the changeFields doesn't contain the include list and there is at least one changed field that finds a match in the excluded changeField's list then the event will be filtered and won't be sent. If no changeFields filter is present, this part of the filter is disregarded and the event will pass this part of the filter. Refer to below table for possible outcomes:

Include List	Exclude List	Result
no list	no list	message sent
no list	no match	message sent
no match	match	message not sent
match	match	message sent
match	no match	message sent
match	no list	message sent

Include List	Exclude List	Result
no list	match	message not sent
no match	no match	message sent
no match	no list	message not sent

changeField example:

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit",
      "changeFields": {
        "include": ["visitId", "lineId", "publishedEta"],
        "exclude": ["publishedEtd"]
      }
    }
  ]
}
```

In the above example, this filter only consists of the changeFields filter. The changeFields filter needs to be written as a map from "include" and "exclude" to their lists of values. In this example the message payload will only be passed through if at least one changed field is "visitId", "lineId", or "publishedEta" and no changed fields are "publishedEtd".

It is possible to only create and excluded list or included list:

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit",
      "changeFields": {
        "exclude": ["publishedEtd", "publishedEta"]
      }
    }
  ]
}
```

This example only filters out events that had the "publishedEta" or "publishedEtd" change.

Combining changeField and predicate Filtering

When more than one filter criteria is present, then all criteria must pass for the overall filter to allow the message through.

This example combines the eventName, changeFields, and filter predicates together to form a VesselVisit event filter that only passes through if the "publishedEta" or "publishedEtd" were changed and the vessel type is "CELL"

```
{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit",
      "changeFields": {
        "include": ["publishedEta", "publishedEtd"]
      },
      "filter": {
        "predicate": ["vesselType=CELL"]
      }
    }
  ]
}
```

```

    }
  }
]
}

```

The above subscriber has one filter, which has 3 conditional filter criteria. Each part of the present criteria will need to return true for a message to pass.

Filtering with Advanced Predicates via JSON Path Filter Expressions

For conditional predicates that need AND(&&) or OR(||), JSON path filter expressions can be used.

These expressions can be placed in the filter's predicate list as is. They will not need the "\$.payload.content" prefix, as described earlier in this developer guide, this prefix will automatically be accounted for.

Example: Want to filter for VesselVisits that have a visit phase (visitPhase) of 60DEPARTED or 40WORKING. The simple predicates from earlier in this developer guide will not be sufficient for this use case, because having a filter with two predicates of

"visitPhase=60DEPARTED" and "visitPhase=40WORKING" will filter everything out. Either one or the other will consistently fail and all messages will be filtered out. The below predicate is the correct way to achieve this use case:

Json Path Filter Expression **for** use **case** of: (visitPhase) of 60DEPARTED or 40WORKING
`[?(@.visitPhase=="60DEPARTED" || @.visitPhase=="40WORKING")]`

Notify Subscriber Post payload:

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit",
      "filter": {
        "predicate": [
          "[?(@.visitPhase=="60DEPARTED" || @.visitPhase=="40WORKING")]"
        ]
      }
    }
  ]
}

```

Explanation:

- 1) Note that the entire filter expression `"[?(<expression>)]"` goes into the predicate.
- 2) Note the escape characters **for** the **double** quotations around `"60DEPARTED"` and `"40WORKING"`, these are necessary because the predicate is a string.
- 3) This filter expression will be **true** or **false**, either the visitPhase will be `"60DEPARTED"` or `"40WORKING"`, which results in the filter passing or failing.

Example: Two or more simple predicates could also be combined to form one advanced predicate. Recall that all predicates in the filter must be true for the predicate criteria to pass. We can replicate this logic with filter expressions and the AND (&&) condition.

This is from a previous example on simple predicates:

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit"
      "filter": {
        "predicate":["visitPhase=60DEPARTED", "vesselType=CELL", "lines[*].lineOutVoyNbr=52"]
      }
    }
  ]
}

```

rewritten with JSON path filter expressions:

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",

```

```

"filters": [
  {
    "eventName": "VesselVisit"
    "filter": {
      "predicate": [
        "[?(@.visitPhase=="60DEPARTED" && @.vesselType=="CELL")]",
        "lines[?(@.lineOutVoyNbr == \"52\")]"
      ]
    }
  }
]
}

```

Explanation:

- 1) The "visitPhase=60DEPARTED" and "vesselType=CELL" can be combined into the single filter expression.
- 2) The "lines[*].lineOutVoyNbr=52" could not be combined into that filter expression, recall from the JSON path filter expression denotes the current element being processed. When comparing lines, which has child elements, the depth of This predicate set up still is correct as all predicates need to **return true** (there is an implicit conditional AND)
- 3) Simple predicates are better **for this use case**, just showing the possibility with conditional AND.

Combining changeFields Filter and Advanced Predicates

Use Case: Want to only view VesselVisits that had a change in the "publishedEta" or "publishedEtd" fields and the vessel is either of type CELL1 or CELL2.

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "transport": "ws",
  "groupId": "sampleGroupId",
  "filters": [
    {
      "eventName": "VesselVisit",
      "changeFields": {
        "include": ["publishedEta", "publishedEtd"]
      }
      "filter": {
        "predicate": ["[?(@.vesselType=="CELL1" || @.vesselType=="CELL2")]" ]
      }
    }
  ]
}

```

Explanation:

This subscription has a filter on VesselVisit events, where **if** any incoming event had a change on publishedEta or publishedEtd

Modifying Payload Content

The filters described in previous sections are only about filtering out messages on certain conditions. It is also possible to modify the message itself. Field filtering with the "fields" filter allows for the content of the payload to be modified. Field elements in the payload content can be chosen to be shown, the fields that are not chosen are omitted from the payload to the subscription.

To choose the desired fields, add the desired fields as a list to the "fields" portion of the filter object within the subscriber.

Continuing with the VesselVisit event sample payload, copied from above:

VesselVisit Event

[Expand source](#)

Example: Only want the VesselVisit's visit ID, lines' lineOutVoyNbr, lineInVoyNbr, lineId, and the estimated shift and restow counts.

JSON posting:

```

{
  "type": "subscribe",
  "channel": "/nsp",
  "groupId": "sampleGroupId",
  "transport": "ws",
  "filters": [
    {
      "eventName": "VesselVisit",
      "filter": {
        "fields": [
          "visitId",
          "lines.lineOutVoyNbr",

```

```

        "lines.lineInVoyNbr",
        "lines.lineId",
        "estMoveCount.estRestow",
        "estMoveCount.estShift"
    ]
    }
}
]
}

```

Result when VesselVisit data gets filtered:

```

"content": {
  "visitId": "DOB196",
  "estMoveCount": {
    "estRestow": 14,
    "estShift": 10
  },
  "lines": [
    {
      "lineOutVoyNbr": "52",
      "lineInVoyNbr": "51",
      "lineId": "MAT"
    },
    {
      "lineOutVoyNbr": "50",
      "lineInVoyNbr": "55",
      "lineId": "MAT"
    }
  ]
}

```

Explanation:

- 1) This Notify Subscription, contains one filter, which is subscribing to VesselVisits. There are no other filter p
- 2) Note that "\$.payload.content." is not needed as a prefix to any of our fields. This will automatically be added.
- 3) Note that the syntax **for** the lines fields only use the dot-notation (.) without brackets. Even though lines is a
- 4) The dot-notation (.) should be used **for** all desired child-fields, like estMoveCount.estRestowCount.
- 5) If the fields filter is empty, or is not present, then the original payload is passed through.

Child Entity Use Case

Child entities in the NSP environment refer to business entities in N4 that are used within another entity.

For example the lines of a vessel visit is noted as a vessel_visit_line entity. This vessel_visit_line entity is a child entity belonging to the parent VesselVisit entity.

On any change to the child entity, the NSP payload only consists of the child entity itself. There is a reference to the parent entity, but the parent entity's state is not provided in the payload.

For the case of notify-subscribers and notify-filters which are filtering over the parent entity, changes to their child entity will not be filtered at a predicate or field level. If there is a subscription to its' parent entity type, the child entity will be sent "as-is".

For example:

Child Entity chagne Vessel Visit Line

```

{
  "businessEntity": "VesselVisit",
  "childAffected": "lines",
  "operation": "update",
  "changes": [
    {
      "field": "lineInVoyNbr",
      "old": "TEST",
      "new": "NEW_TEST"
    }
  ],
  "payload": {
    "keys": [
      {
        "scope": {
          "yard_gkey": null,
          "yard_id": null,
          "facility_gkey": 1,
          "facility_id": "FCY111",

```

```

        "complex_gkey": 1,
        "complex_id": "CPX11",
        "operator_gkey": 1,
        "operator_id": "OPR1",
        "instance_id": "test_instance",
        "level": "FACILITY"
    },
    "entityName": "VesselVisit",
    "id": "BEST1000",
    "gkey": 4
}
],
"childKeys": [
{
    "scope": {
        "yard_gkey": null,
        "yard_id": null,
        "facility_gkey": 1,
        "facility_id": "FCY111",
        "complex_gkey": 1,
        "complex_id": "CPX11",
        "operator_gkey": 1,
        "operator_id": "OPR1",
        "instance_id": "test_instance",
        "level": "FACILITY"
    },
    "entityName": "VesselVisitLine",
    "id": "BEST1000",
    "gkey": 57
}
],
"content": {
    "lineOutVoyNbr": "TEST",
    "lineInVoyNbr": "NEW_TEST",
    "lineId": "APL",
    "cargoCutoff": null,
    "emptyPickup": null,
    "beginReceive": null,
    "reeferCutoff": null,
    "hazCutoff": null
}
}
}

```

This is the resulting payload of a change to the lines child entity of VesselVisit. Note the content portion of the payload only consists of the child entity's state.

This payload will be forwarded to the desired subscription endpoint if the subscription is accepting VesselVisit events. Presently, with exception to the "businessEntity" field, child events will not be filtered for predicates or fields.

Instance Security in Rest API

If instance security required , Smart application may include one or more scoped business units in API request as part of '**predicates**' query parameter.

If scoped business units are not provided, entities affiliated with all companies are included in the response.

Instance security filter will be applied if:

- Instance Security is enabled
- Entity class or field is annotated with @InstanceSecurityFields and/or @InstanceSecurityField
- Smart application includes includes one or more scoped business units in API request

Predicate Name

- bizroles

Predicate Value

- Base64 encoded Map<BizRoleEnum, Set<Long>>

① Example biz company roles request is {"LINEOP":[1],"HAULIER":[11,22,33]}

Since **predicates** query parameter in NSP is comma delimited list of key value pairs representing predicates, commas in biz roles would clash with predicate separator.

Easiest solution was to Base64 encode it.

Example

Example how to format URL bizrole predicate value
<pre>String predicateJson = "{\"LINEOP\":[1],\"HAULIER\":[11,22,33]}" String bizRolePredicate = Base64.getEncoder().encodeToString(predicateJson.getBytes()); // bizRolePredicate = eyJMSU5FT1A101sxxX</pre>
Example of URL query parameter
<pre>http://\${host}:\${port}/inventory/workinstructions?operator=OPR1&predicate=ownerPowGkey:3,bizroles:eyJMSU5FT1A101sxxX</pre>

Rest API

NSP does not manage user association with business units, though it provides API to retrieve N4 user affiliations managed in N4 itself.

Business Units by Role Rest Endpoint

Returns affiliated companies for N4 users managed in N4.

Request

Endpoint

- GET /users/businessunitsbyrole

Query Parameters

- userid (required) - N4 User Id
- instanceid (required) - N4 instance id

Example

- http://\${host}:\${port}/users/businessunitsbyrole?userid=User1&instanceid=navis_dbz

Response

Map of BizRoleEnum to set of company gkeys - Map<BizRoleEnum, Set<Long>>

Example

- {"LINEOP":[1],"HAULIER":[11,22,33]}