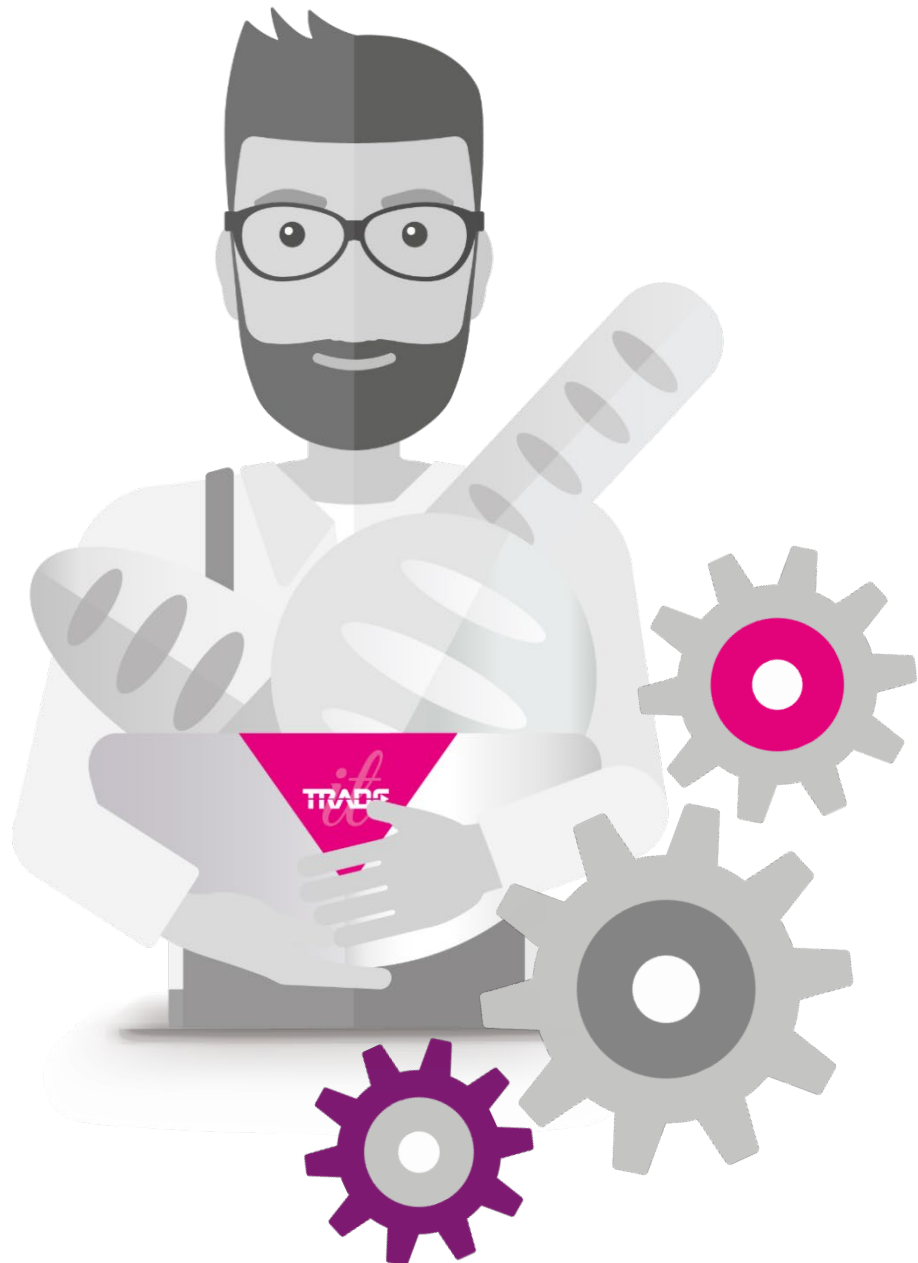


Tradesolution EPD

A quick guide to the EPD API



Tradesolution © 2018

Document version 1.5

Innhold

1.0	Change log	4
2.0	About this document.....	5
3.0	What is EPD?.....	5
4.0	STAND011.....	6
5.0	EPD Standard 10	6
6.0	Registration of product information in phases	7
6.1	The registration processes.....	7
7.0	Suppliers and recipients	8
8.0	Product and units	8
8.1	A product is a hierarchy of units.....	9
8.2	Unmarked variants	13
8.3	Summary.....	14
9.0	How to get product information?	15
10.0	How to upload product information?.....	15
10.1	From draft to product (upload and validation process)	15
10.2	How to create an empty draft (kladd)?	15
10.3	How to create a draft with data?	16
10.4	Update an existing draft	17
10.5	Update an existing product	17
11.0	Lookup tables	18
12.0	Error labelling	18
13.0	Messages	18
14.0	Events	18
14.1	Events that can be subscribed to	20
15.0	Authentication.....	23
Appendix 1 – How to get response from swagger		24
Appendix 2 – the product registration process.....		25

1.0 Change log

Version 1.6:

Added tips for how the subscriber of events should implement to the events in section 14.

Version 1.5:

Updated technical description of the events in section 14.

Version 1.4:

Update descriptions on the events in section «Events». Added «Appendix 1 – How to get respons from swagger». Added «Appendix 2 – the product registration process» with examples from the product registration and QA-process.

Version 1.3:

Added descriptions on how to upload drafts and edit products in section «How to upload product information».

Version 1.2:

Added example of basis and mellom units with multiple topps. See paragraph «Product and units».

Version 1.1:

Added functionality to handle «unmarked variants». See paragraph «Product and units».

Version 1.0:

Document renamed. Added the paragraph «How to upload product information?»

2.0 About this document

This document is a quick guide to getting started with the EPD API which was launched in 2018. The complete technical description of the REST API is documented in Swagger.

See: <http://epdapi.tradesolution.no/swagger/>

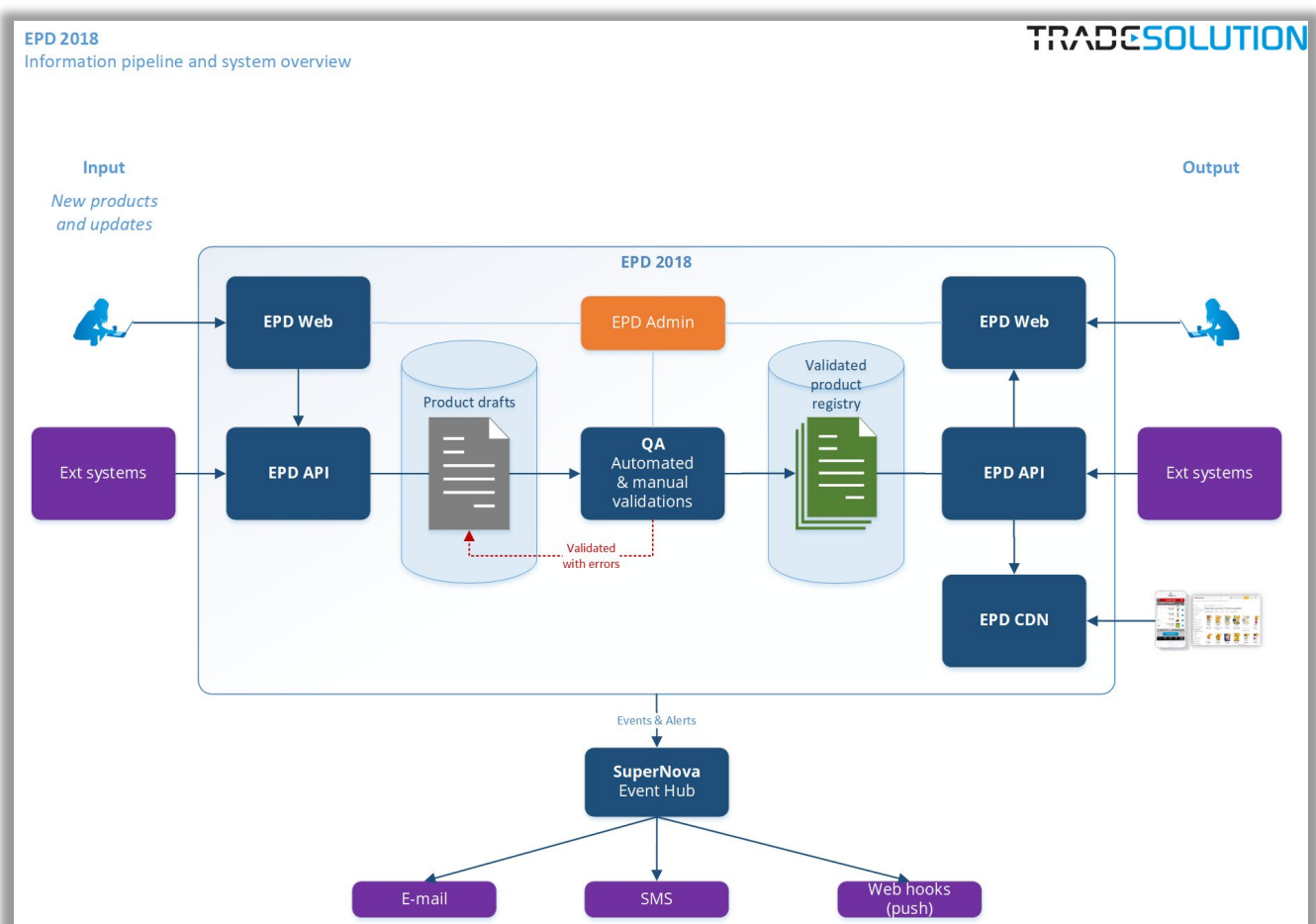
If more information is needed contact epd@tradesolution.no.

3.0 What is EPD?

EPD is a master data management system for the Norwegian grocery segment. EPD gives access to product data to its recipients (subscribers of data) such as wholesalers, grocery chains and web shops.

The suppliers (product owners) are responsible for having complete and correct product data of all the products they offer in the grocery segment. EPD have an extensive quality assurance system to help them providing correct data.

No product data is available for the subscribers before it has passed an extensive validation service (the Quality Assurance service – hereafter called QA).



For more information of EPD: www.tradesolution.no/tjenester/epd/

4.0 STAND011

EPD are based on principles from STAND011 which states the responsibilities between suppliers and retailers during the implementation of new products to the Norwegian grocery market.

The registration of new products i

s set to three default launch windows. Each launch window has a described process and responsibilities.

For more information: www.stand.no/standarder/stand011.

5.0 EPD Standard 10

The EPD API is based on the required information for a product (the product catalog) described in EPD Standard 10.

The EPD Standard 10 is described in detail in the document «EPD_Standard_10.xlsx». See www.tradesolution.no/tjenester/epd/.

Active standards in EPD can be listed by the API call [Standard](#).

6.0 Registration of product information in phases

The new EPD Standard 10 divides the registration of the required product information in sequential phases with individual deadlines. The overall process is referred to as a launch window (see STAND011).

This is a major shift from the previous standards of EPD where all required product information had to be registered before a single deadline.

The EPD Standard 10 also require interaction from selected recipients.

The deadline of each phase is set to a given number of weeks before the product should be launched in the stores. The launch date is defined as week '0'.

See overview of the registration process below.

The required product information for each phase are described in detail in EPD Standard 10.

The example below shows the phases prior to the launch date (week '0'):



6.1 The registration processes

► EPD F1 (phase 1) week -15

The deadline to the first registration phase is set to 15 weeks before launch (week -15). The required information needed for this phase must be registered and validated (passed QA) within the end of week -15. The status of the product is set to: «Godkjent F1» in EPD and an event is sent to the event system (see paragraph «Events» for more information). Responsibility: Supplier

► EPD F1 Listing (phase 1 listing) week -8

Selected retailer gives information of which new products registered by the suppliers in EDP F1 they want to sell (give listing to). The retailers must post the selected products using the POST method of [Produkt/IncludeInSortiment](#) within the end of week -8. When the post is done, the event is sent to the event system. Responsibility: Selected retailers

Note: The retailers can also post products that have been registered in previous launch windows with status «Godkjent F1» or «Godkjent F2» in EPD.

► EPD F2 (phase 2) week -6

The supplier must have registered and validated (passed QA) the product information as required in EDP F2 within end of week -6. The product will then get status «Godkjent F2» in EPD and the event is sent to the event system. Responsibility: Supplier

► EPD Sjekkpunkt (phase 3) week -3

The products that are chosen to be listed in EPD F1 Listing must be sent to Tradesolution Sjekkpunkt for physical controls within week -3. If the product passes the control a «checked date» (SistKontrollmaalt) is set for the controlled unit. Responsibility: Supplier

Information of the phases used in EPD can be listed by this API call [Standardfase](#).

You will find more information on the product registration process in «Appendix 2»

7.0 Suppliers and recipients

Both suppliers and recipients are identified by GLN (Global Location Number).

- ▶ A recipient can get access to product information from suppliers who have registered their products in EPD
- ▶ A recipient can only get information from the suppliers they have subscribed to
- ▶ A supplier can deny a recipient access to one or many of their products
- ▶ In EPD a supplier is called «produkteier»
- ▶ In EPD a recipient is called «mottaker»

API call to see all suppliers: [Produkteier](#).

API call to see all recipients: [Mottaker](#).

If a recipient wants to subscribe to products from a new supplier, it is possible to send «a request for information» call to the supplier. See [ProdukteierRfi](#) (documentation not available).

The supplier can then grant or deny the request.

8.0 Product and units

A product in EPD consists of a set of units in a hierarchy. A product is identified by a EPD number (EPDnr) and a unit is identified by GTIN (Global Trade Item Number). All EPD numbers and GTIN's are connected to a supplier identified by GLN.

Be aware of that two suppliers can sell the same unit (GTIN) with different EPDnr. The unit (GTIN) will be registered by both parties. The combination of GTIN+GLN are always unique.

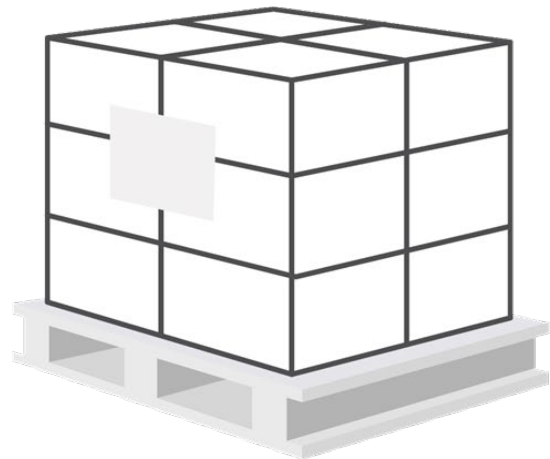
Three base units (basis) often referred to as customer units:



Three «mellom» units often referred to as trade units:



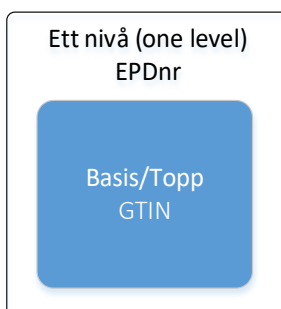
Examples of «topp» units often referred to as distribution units:

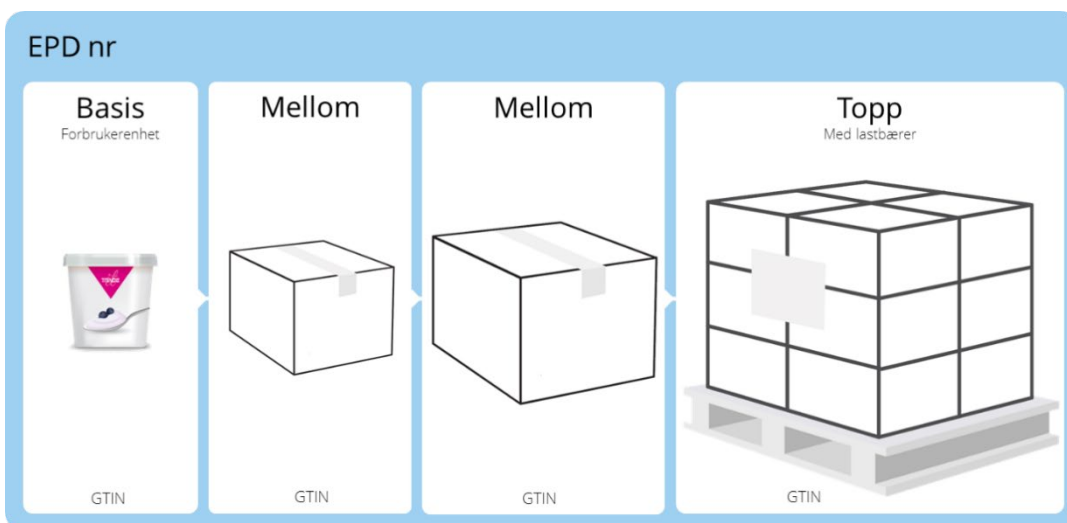
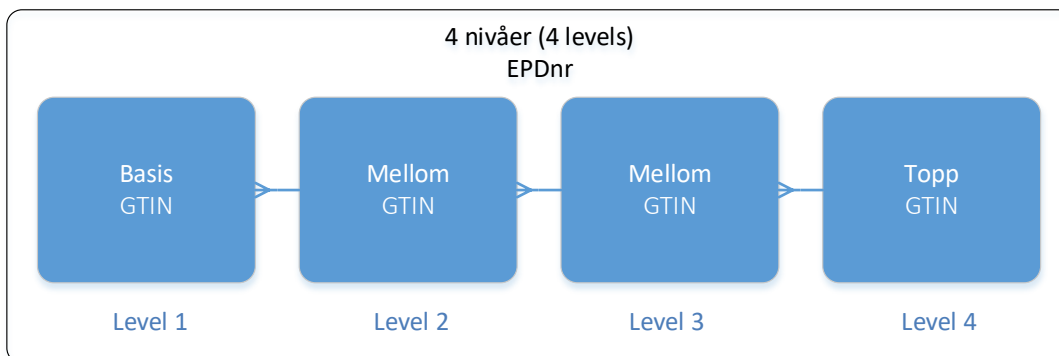
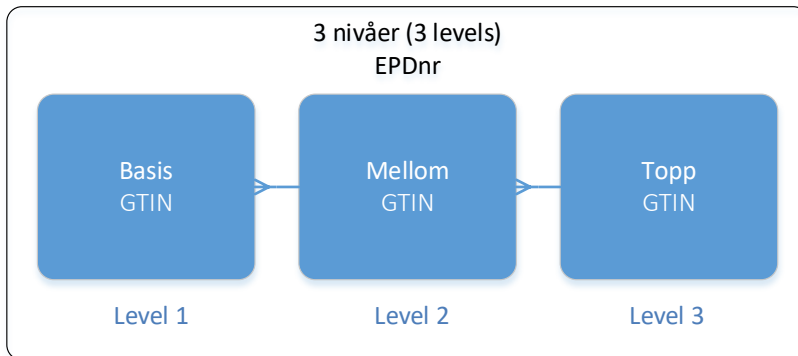
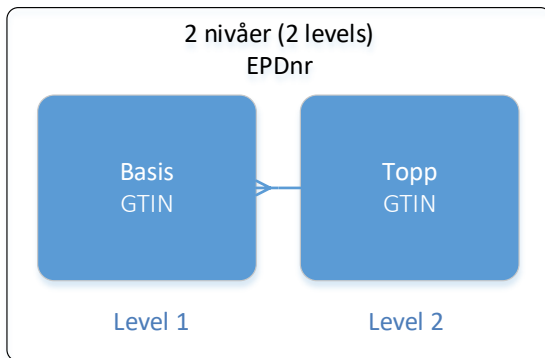


8.1 A product is a hierarchy of units

To create the relation between two units in a product the smaller unit in a product have a reference to the unit above.

A hierarchy can have up to 4 levels. See the examples below of products with different levels of units (note the naming of each level):

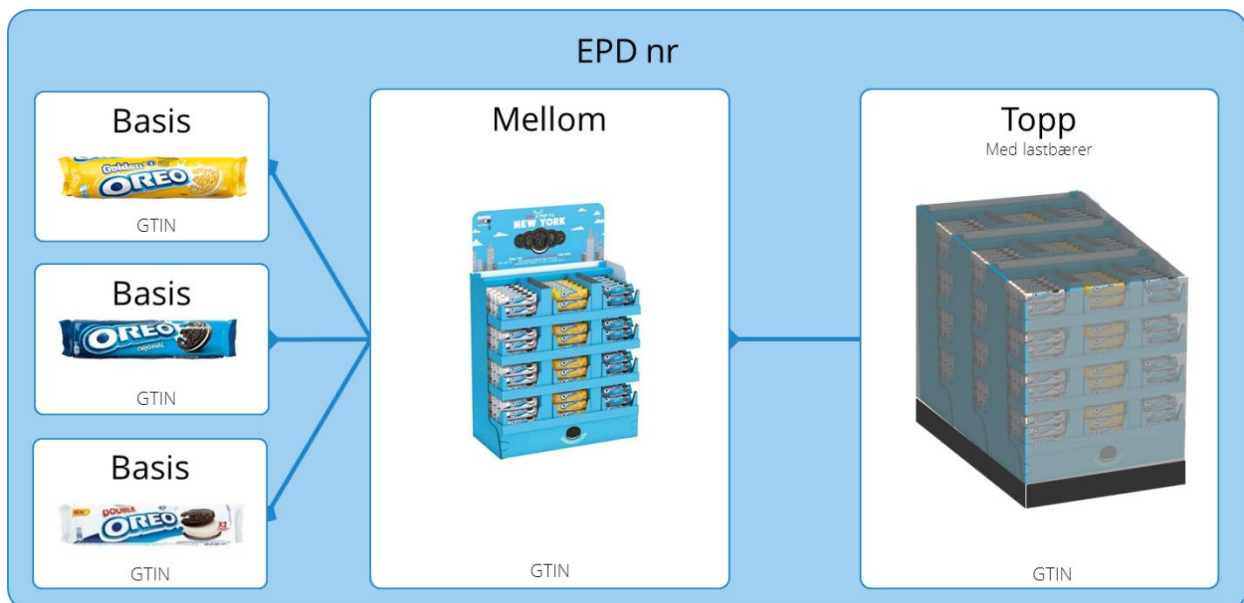
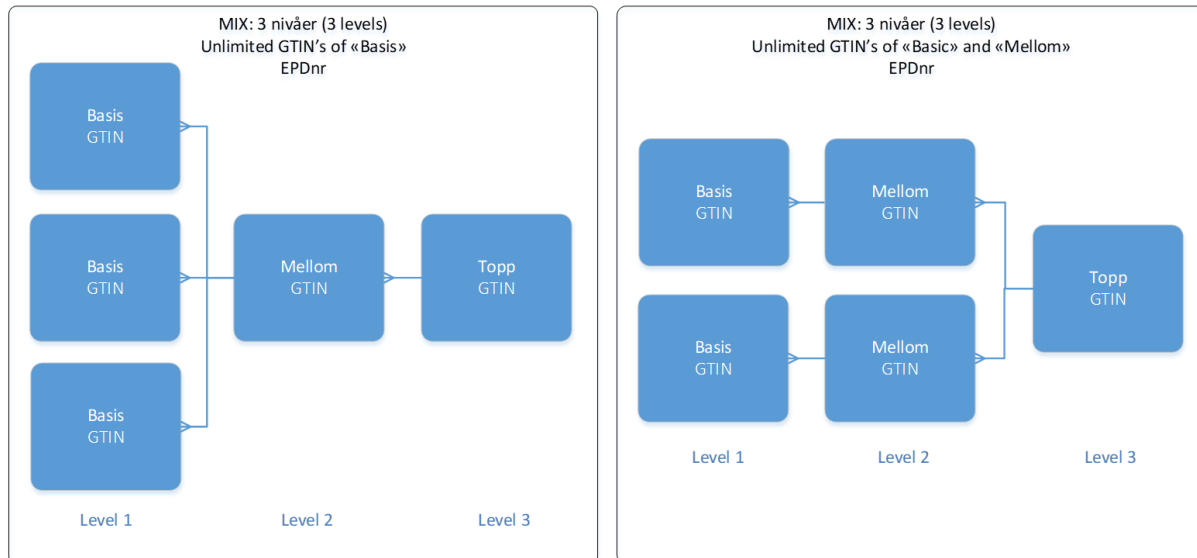




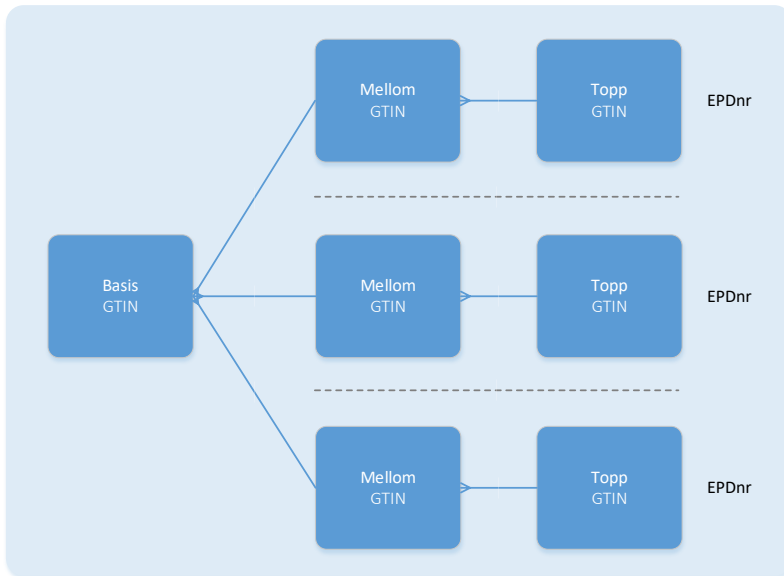
Note

One of the units (normally «mellom» or «topp») will be marked as the purchase unit from the supplier.

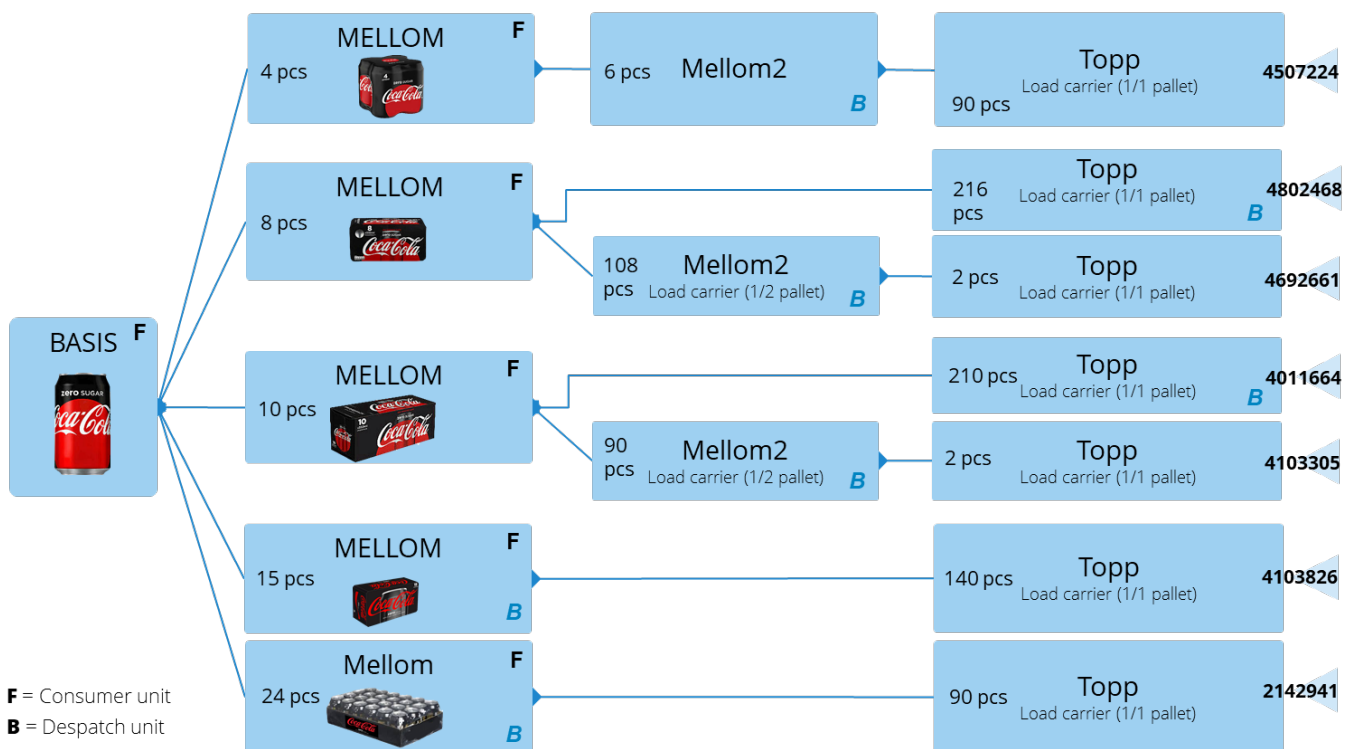
A product can also have multiple base (basis) units or trade (mellom) units. See the examples below:



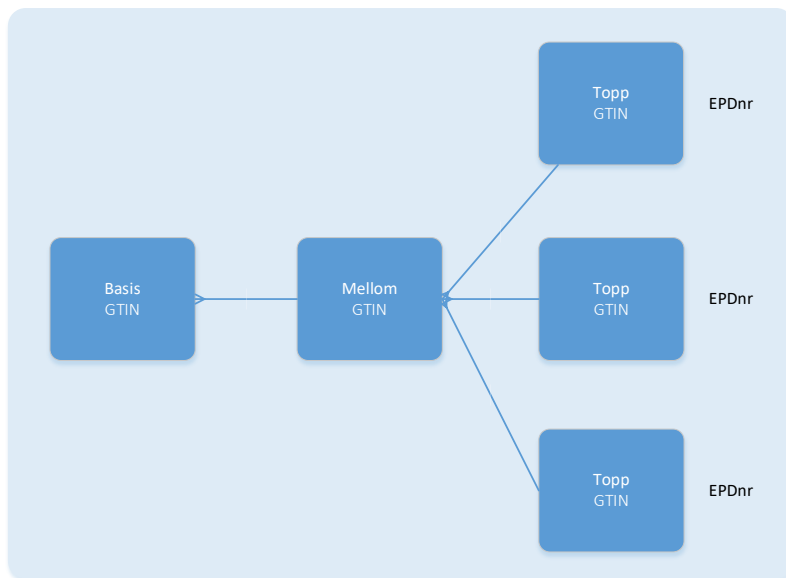
A base unit (basis) unit can also be included in many product hierarchies (many products will have the same base unit – a base unit can be part of many EPDnr). See the examples below with three different products (EPDnr) using the same base (basis) unit:



Example with Coca Cola Zero. The numbers to the right are actual EPD# (active products per January 2018):



A basis and mellom unit can also be part of multiple «topps»:



IMPORTANT: Phases in hierarchies

An upper level unit cannot be upgraded to phase 2 if the lower level is in phase 1.

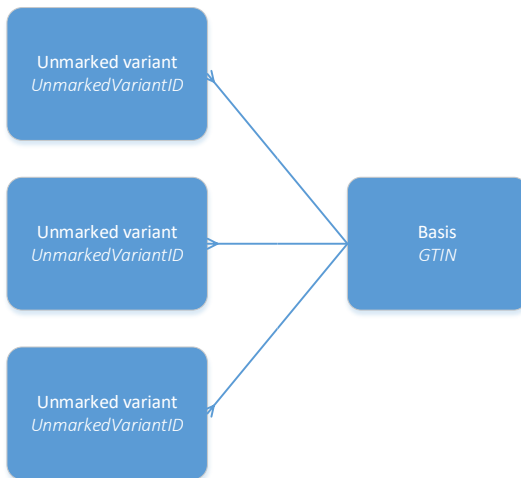
8.2 Unmarked variants

A basis unit can contain units without identification. The «unmarked variant» is not labelled with a barcode (GTIN) and can therefore not be sold separately. If there are several unmarked variants EPD will give them an ID («Umerketvariantnr») starting with '1' and counting for each unmarked variant in the basis unit (GTIN). The «Umerketvariantnr» can only be used together with the GTIN of the basis unit.

See example below of a product which have four different unmarked variants which cannot be sold separately and have no barcode (GTIN):



The basis level can include several «unmarked variants» identified by “Umerketvariantnr”:



8.3 Summary

- ▶ A product is identified by a EPD number (EPDnr)
- ▶ A unit is identified by a GTIN (Global Trade Item Number)
- ▶ A unit is called «pakning» in EPD
- ▶ A product can have up to 4 levels of units (hierarchy)
- ▶ A product can be a mix of multiple «basis» or «mellom» units
- ▶ A “basis” unit can be shared by many products
- ▶ The first level of the units is called «basis» in EPD (often referred to as a customer unit or base unit)
- ▶ The top level of the units is called «topp» in EPD (often referred to as a distribution unit)
- ▶ The levels between «basis» and «topp» (if any) are called «mellom» units
- ▶ A basis unit can consist of units without identification called unmarked variants («umerkede varianter»)
- ▶ An upper level unit cannot be upgraded to phase 2 if the lower level is in phase 1

9.0 How to get product information?

API request related to access product information is PRODUKT (product) and PAKNING (unit).

PRODUKT give all the product information included units.

PAKNING give information on single units.

See [Produkt](#) and [Pakning](#).

10.0 How to upload product information?

10.1 From draft to product (upload and validation process)

TIPS: You will find more information on the product registration process in «Appendix 2»

To create a product (produkt) in EPD you need to start by creating a draft (kladd).

The draft must be valid according to a specific standard (for 2018 it is «EPD Standard 10»). When the draft is created according to the standard you must then send it to the QA-service (Quality Assurance). QA will then validate the draft. If the draft passes the validation the draft will be approved. An approved draft will get a unique EPD number (EPD nr). The draft is then converted to a product that is immediately available for the receivers. If the validation fails an error message will be posted.

There are two ways to create a new product. One way is to create an empty draft (described in 10.2). and the other is to create a draft with data (described in 10.3).

10.2 How to create an empty draft (kladd)?

The EPD API provides several post methods from which you can create an empty draft.

When creating an empty draft, the structure of the product is created in the database along with the standard it belongs to.

The methods you can use to create an empty draft are described below. All these methods return a «KladdProduktID». «KladdProduktID» is a unique number that identifies the newly created draft.

Api metode (POST)	Description
/api/Kladd/Create_B	Create kladd with only a single pakning.
/api/Kladd/Create_B_T	Create kladd with new basispakning and new toppakning
/api/Kladd/Create_B_M_T	Create kladd with new basispakning, new mellompakning and new toppakning
/api/Kladd/Create_B_M_M_T	Create kladd with new basispakning, new level-1 (mellom1) mellompakning, new level-2 (mellom2) mellompakning and new toppakning
/api/Kladd/Create_ExistingB_T	Create kladd with an existing basispakning and new toppakning. The existing basispakning must be in an approved product.
/api/Kladd/Create_ExistingB_M_T	Create kladd with an existing basispakning, new mellompakning (mellom1) and new toppakning.

	The existing basispakning must be in an approved product.
/api/Kladd/Create_ExistingB_M_M_T	Create kladd with an existing basispakning, new level-1 mellompakning (mellom1) , new level-2 (mellom2) mellompakning and new toppakning. The existing basispakning must be in an approved product.
/api/Kladd/Create_ExistingM_T	Create kladd with an existing mellompakning (mellom1) and new toppakning. The basispakning connected to the mellom1 will also be included in the new kladd. The existing mellompakning must be in an approved product.
/api/Kladd/Create_ExistingM_M_T	Create kladd with an existing level-1 (mellom1) mellompakning, new level-2 (mellom2) mellompakning and new toppakning. The existing mellompakning must be in an approved product.
/api/Kladd/Create_Mix_Minimum2B_T	Create kladd with minimum 2 basispakninger and new toppakning (mix). If existing basispakning is selected it must be in an approved product.
/api/Kladd/Create_Mix_Minimum2B_M_T	Create kladd with minimum 2 basispakninger, new mellompakning (mix) and new toppakning. If existing basispakning is selected it must be in an approved product.
/api/Kladd/Create_Mix_Minimum2B_M_M_T	Create kladd with minimum 2 basispakninger, new level-1 (mellom1) mellompakning (mix), new level-2 (mellom2) mellompakning and new toppakning. If existing basispakning is selected it must be in an approved product.
/api/Kladd/Create_Mix_Minimum2M_T	Create kladd with minimum 2 mellompakninger (mellom1) and new toppakning (mix). The selected mellompakninger can be in an existing and approved produkt, or new mellompakning with one or more selected basispakninger.
/api/Kladd/Create_Mix_Minimum2M_M_T	Create kladd with minimum 2 mellompakninger (mellom1), mellom2 pakning (mix) and new toppakning. The selected mellompakninger can be in an existing and approved produkt, or new mellompakning with one or more selected basispakninger.

Mentioned earlier, the returned “KladdProduktID” is the unique identifier for the created product. This KladdProduktID must be used when updating the kladd.

Read more details about these methods in swagger. See [Kladd](#).

10.3 How to create a draft with data?

In some situations, you need to create a new draft with data. The situations can be:

- a) You have your own system with product data and needs to transfer that data to EPD

b) You have already an approved product in EPD and need to make some changes to the product

To create a new draft with data, use these methods:

Api method (POST)	Description
/api/Kladd/Create	<p>Create new kladd with data. A new kladd will be created only if structure of request.Kladd is a valid structure.</p> <ul style="list-style-type: none"> • Data in request.Kladd will be copied to created kladd • Validation is executed if request.Validate is set to true • Kladd will be sent to quality assurance if request.SendToQA is set to true and kladd is valid <p>If you try to create a kladd with EPDnr and that EPDnr already exists in a kladd, then no kladd will be created.</p>
/api/Kladd/EditProdukt	Create kladd for an existing product. Notice that there can only exist one kladd for each product.

Read more details about these methods in swagger. See [Kladd](#).

10.4 Update an existing draft

There are two ways to update an existing draft

- Update existing draft by posting the full «kladdProdukt» object. In general all the draft data that is saved in the database will be replaced by the data in the «KladdProdukt» object. Notice that in some cases the data will not be replaced, or in other cases some of the data will not be replaced. See swagger documentation for more details.

Use the following API post method:

```
> /api/Kladd
```

- Update existing draft by posting/patching the data that has been changed. You do not have to post the entire «KladdProdukt» object to update the data in the database. When only one field has changed you are able to post the changed value for the field only.

Use the following API patch method to do a partial update:

```
> /api/Kladd/Patch
```

10.5 Update an existing product

- To update the existing product a draft must be created. Only one draft can exist for each product. A product can be edited for following purposes:
 - 1) Move a product from Fase1 to Fase2.

- 2) Change information for a Fase1 product
- 3) Change information for a Fase2 product

Use the following API post method:

```
> /api/Kladd/EditProdukt
```

11.0 Lookup tables

You can also use the EPD API to get all the information in the various lookup tables of the EPD system. The requests are marked with «LOOKUP» in the description in Swagger.

More information in «EPD_Standard_10.xlsx».

12.0 Error labelling

The system allows a receiver to post an «error label» for specific product information. The error label can contain a comment and a suggestion for a new value.

See [ProduktFeilmerking](#).

13.0 Messages

The EPD system facilitate a message dialog between supplier and receiver for a specific product. The messages from a supplier can be routed to all receivers of the product or to a specific receiver. The messages from a receiver will be routed to the supplier of the product.

See [ProduktDialog](#)

and [ProduktDialogMelding](#).

14.0 Events

Events from EPD can be subscribed to by the recipients. Events will then be pushed to a specific URL when they occur. This is achieved by webhooks. A WebHook is an HTTP callback: an HTTP POST that occurs when something happens. The events will be posted as JSON on the following format:

```
{
  "EventID": "UniqueID",
  "Event": "name_of_event",
  "Source": "nameOfSystemThatSentEvent",
  "Data": {},
  "Timestamp": "whenEventWasSent",
  "Text": "eventInTextFormat_ForUseWithTeamsOrSlack"
}
```

That means that an event object can be posted, and each event can contain different payloads. An example event could look like this:

```
{
  EventID: "19454b90-b9b7-4c29-90d0-0e79766e481a"
  Event: "produkt_godkjent_fase1",
  Source: "EPD",
  Data:
  {
    "EpdNummer": 1111111,
    "KladdProduktID": 2222222
  },
  "Timestamp": "2018-02-08T12:22:40.7542409+00:00",
  "Text": "Event = Produkt godkjent i fase 1, Source = EpdPortal, Payload = \r\n
EpdNummer: 222222\r\n\r\n"
}
```

It is important to design the system that receives a webhook call as idempotent. An idempotent operation is one that has no additional effects if it is called more than once with the same input parameters. If two events with the same payload is sent, the receiving system should end up in the same state after both have been received. The same holds true if these events trigger information retrieval (I.e. a get request to EPD API) and the same information is returned both times. However, if data in EPD has changed during the time between the two calls, the calling system should end up in different states after each call.

The preferred reception method is to store or queue the received message and then return success (200 OK). Further processing should occur independently from the webhook call. If a webhook is called with an event that it currently does not support or handle, it should ignore it (and ideally log it internally) and return success (200 OK).

Default security for webhook calls is HTTPS against the URL specified. The endpoint can use a «secret random» URL and/or a query parameter with a secret. This secret can also be included as a custom header. Some examples are:

Secret url as used by Microsoft Teams:

<https://outlook.office.com/webhook/skjh546kiluh sdf87yjh5b4uidf87/IncommingWebhook/lihkj6547nsdaf78oykj bdfq/lkjin asfq987iu45eio ljsadf987ys>

Secret in query parameter as used in Azure Functions:

<https://functions.azurewebsites.net/api/TestWebhook?code=kjhadfsh4w56kj huds f8745lkisdfg nkjm iouwert qsd f q==>

If firewall exceptions are required, use IP address 52.174.3.80

14.1 *Events that can be subscribed to*

▶ **produkt_godkjent_fase_1**

Occurs when a product is validated and approved for EPD F1 by QA. The product status is set to «Godkjent F1». The event returns the EPDnr and KladdProduktID for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_lagt_til_listing**

Product added to listing from a subscriber. The event will be sent to the product owner.

▶ **produkt_godkjent_fase_2**

Occurs when a product is validated and approved for EPD F2 by QA. The product status is set to «Godkjent F1». The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers. The event will be sent to both the product owner and the subscribers.

▶ **produkt_godkjent_sjekkpunkt**

Occurs when a product is validated and approved by EPD Sjekkpunkt. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_godkjent_sjekkpunkt_med_endringer**

Occurs when a product is validated and approved by EPD Sjekkpunkt with edits/updates. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **kladd_feilmerket**

Occurs when a draft is error-marked by QA. The event returns the draft ID for the product in the payload and the phase. The event will be sent to the product owner.

▶ **produkt_feilmerket_fase_1**

Occurs when a product in EPD F1 is error-marked by a receiver (mottaker). The event returns the EPDnr for the product in the payload.

▶ **produkt_feilmerket_fase_2**

Occurs when a product in EPD F2 is error-marked by a receiver (mottaker). The event returns the EPDnr for the product in the payload.

▶ **produkt_feilmerket_sjekkpunkt**

Occurs when a product is error labelled by EPD Sjekkpunkt. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_feilmerking_trukket**

Occurs when a receiver (mottaker) withdraw an error-marking. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_feilmerking_rettet**

Occurs when a supplier (produkteier) fixes an error. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_feilmerking_avvist**

Occurs when a supplier (produkteier) rejects an error-marking by a receiver (mottaker). The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the subscribers.

▶ **produkt_dialogmelding_sendt**

Occurs when a supplier (produkteier) or a receiver (mottaker) creates a new dialog on a specific product. Only the involved receiver (mottaker) and the supplier (produkteier) will receive this event. The event returns the EPDnr for the product in the payload. The event will be sent to both the product owner and the selected subscriber.

▶ **sokt_om_produktendring**

Occurs when a supplier (produkteier) is requesting to change a locked field (that requires an application if to be unlocked) in a product. The event returns the EPDnr for the product in the payload.

▶ **produkt_endret_gtin**

Occurs when a supplier changes a GTIN for a specific unit

▶ **produkt_endret_antall**

Occurs when a supplier changes "antall" for a specific unit

▶ **produkt_endret_mengde**

Occurs when a supplier changes "mengde" for a specific unit

▶ **soknad_sendt_til_mottaker**

Occurs when a supplier (produkteier) sends an application requesting to change a locked field. The event will be sent to both the product owner and the selected subscriber.

▶ **soknad_godkjent_av_mottaker**

Occurs when a subscriber approves the change requested in the application. The event will be sent to both the product owner and the selected subscriber.

▶ **soknad_avvist_av_mottaker**

Occurs when a subscriber declines the change requested in the application. The event will be sent to both the product owner and the selected subscriber.

▶ **soknad_kladd_opprettet**

Occurs when a draft is created, after all necessary subscribers approve the change requested in the application. The event will be sent to both the product owner and the selected subscriber.

▶ **soknad_produkt_godkjent**

Occurs when a draft created by an application is approved by QA. The event will be sent to both the product owner and the selected subscriber

▶ **soknad_trukket**

Occurs when a supplier (produkteier) retracts an application requesting to change a locked field. The event will be sent to the product owner and the selected subscribers.

TIP: You will find more information on how the events are used in the product registration process and QA-process process in «Appendix 2»

15.0 Authentication

EPD API uses OAuth2 bearer token authentication. You can get token by using this POST method:

POST <https://gatekeeper.tradesolution.no/connect/token>

Headers

Content-Type: application/x-www-form-urlencoded

Body

client_id=<insert_client_id>&client_secret=<insert_secret>&grant_type=client_credentials&resource=https://trades.no/TradesolutionApi

Response

```
{
  "access_token": "eyJhbG*****",
  "expires_in": 36000,
  "token_type": "Bearer",
  "scope": "openid profile tradesolutionApi"
}
```

The token is valid for 10 hours.

Add the token to the Authorization header when calling any api endpoint

Appendix 1 – How to get response from swagger

The online api documentation is generated by swagger. Since the api requires authentication, you have to login to get a response directly from swagger.

A brief description of how you can login and get a response:

Produkt

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

GET	/api/Produkt/GetProduktByEPD/{epdnr}	Returns the produkt that matches the epd number
GET	/api/Produkt/GetProdukter	Returns a list of produkt matched by filter
GET	/api/Produkt/GetFerdigeProdukter	Returns a list of produkt that has shared for the client data

- Click the red icon with the exclamation:

Produkt

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)


GET	/api/Produkt/GetProduktByEPD/{epdnr}	Returns the produkt that matches the epd number
-----	--------------------------------------	---

Implementation Notes
 Caller must be a **mottaker** or **produkteier**.
 Caller must subscribe to **produkt/produkteier** or own **produkt** in order to receive the **produkt**.

General info about the produkt object:
 The **produkt** object contains information about a single **produkt**. The EPDnr is unique, and will not appear in any other **produkt** other than the one it is assigned to.
 A standard **produkt** will contain 3 **pakning**, but in some cases less than 3 **pakning**, and some other cases more than 3 **pakning**.
 In the standard **produkt** the first **pakning** will be a "BASIS" (i.e. a jam jar), the second "MELLOM" (i.e. a box containing 9 jam jars) and the third "TOPP" (i.e. pallet containing 12 jam jar boxes)
 All the **pakning** contains the object **Basispakning**, but in a standard **produkt** only the first **pakning** will actually contain any information in **Basispakning** object. In "MELLOM" and "TOPP" the **Basispakning** object will be null.
 The **produkt** object will contain one **pakning**. This **pakning** is the "TOPP" in the **produkt**. The "TOPP" **pakning** can contain a list of **pakning**, and each of these can contain list of **pakning**. The list of **pakning** in a single **pakning** symbolize the **pakning** in a lower level. I.e. a "TOPP" **pakning** can contain one or many "MELLOM" **pakning**, and each of the "MELLOM" **pakning** can contain a "BASIS" **pakning**.
 There are maximum 4 levels of **pakning** (TOPP-MELLOM-MELLOM-BASIS).

Response Class (Status 200)
 Success

Model | Example Value



- You will now be redirected to a login page. Please enter your credentials.
- After successful login the red icon turns green.
- You can now enter epdnr and click on the "try it out!" button:

Parameters


Parameter	Value	Description
epdnr	(required)	

Response Messages

HTTP Status Code	Reason	Response Model
400	Bad Request	Model Example Value

```
{
  "Code": "string",
  "Message": "string",
  "Errors": [
    "string"
  ]
}
```

Try it out!



- Check out the result in the "Response Body". The result is returned in json format.

Appendix 2 – the product registration process

This appendix shows the interaction between the registration phases (product registration), the quality assurance process and events.

