

Flex trading with Capacity Steering Contracts (CSC / CLC) using UFTP messages

Last updated: 07-11-2025

GOPACS supports flex trading with [Capacity Steering Contracts](#) and alternative non-firm transport rights such as TDTR/VVTR through the exchange of Shapeshifter UFTP messages. The implementation must be compliant to the [Shapeshifter UFTP specification](#) version 3.0.0 (for TDTR and VVTR, 3.1.0 is required), with some restrictions which are described separately in this document.

i Note, CSC is the new name for CLC (Capacity Limiting Contracts, NL: CBC / Capaciteitsbeperkende contracten). Where we write CSC, you can also read CLC.

Table of contents

CSC message flow

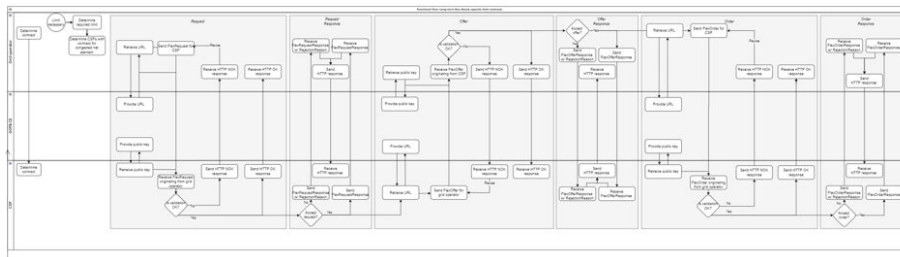
This requires minimum version 3.0.0, but some improvements are made in 3.1.0. Implementations should adopt version 3.1.0 as soon as possible.

Overview of the GOPACS and UFTP message exchange for CSC contracts. The part where the URL and the public key is retrieved, is GOPACS specific.

Currently, a CSC call/activation (*CSC-afroep*) uses the following message flow, where each message request must be first responded to with a 200 OK response before sending back the next message type:

1. Grid operator calculates prognosis, expects flexibility is needed and initiates a call (*afroep*).
2. **FlexRequest** message is sent by the grid operator to the trading company, requesting a certain amount of flexibility.
3. **FlexRequestResponse** message is sent by the trading company to the grid operator, acknowledging the receipt of the request.
4. **FlexOffer** message is sent by the trading company to the grid operator
 - a. The **FlexOffer** must have the same ISPs and capacities as the **FlexRequest**, since in the current implementation, there is no option to actually do an offer and adjust any values of the call.
 - b. The price field is not used yet since all pricing is determined by the contract, there is currently no price negotiation.
5. **FlexOfferResponse** message is sent by the grid operator to the trading company, acknowledging the receipt of the offer.
6. **FlexOrder** message is sent by the grid operator to the trading company, to procure the flexibility.
7. **FlexOrderResponse** message is sent by the trading company to the grid operator, acknowledging the receipt of the order.
8. The call/activation (*afroep*) is now completed.

i We acknowledge that for the current limited use of the UFTP protocol, the full flow of this message sequence would not be needed for flex trading, but the full message flow is implemented to be prepared for the future where offers and price negotiation will most probably be added to the functionalities.



For a readable version of the above diagram, please check the support documents sections.

TDTR/VVTR message flow

This requires version 3.1.0.

Currently, a call (afroep) of TDTR/VVTR contracts uses the following message flow:

1. Grid operator calculates prognosis, expects flexibility is needed and initiates a call (*afroep*).
2. `FlexOrder` message is sent by the grid operator to the trading company, to procure the flexibility.
3. `FlexOrderResponse` message is sent by the trading company to the grid operator, acknowledging the receipt of the order.
4. The call (*afroep*) is now completed.

There is no negotiation phase where a `FlexRequest` and `FlexOffer` are exchanged.



Message broker

The UFTP protocol allows direct communication between grid operators and trading companies. However, for the sake of monitoring and reporting, UFTP messages should be communicated via the GOPACS UFTP Message Broker. The message broker will forward the messages to the specified UFTP recipient. The message broker is does not validate the messages. Validation needs to be done by the recipients themselves.



Open source library

To aid in building a compliant implementation GOPACS has built an open source library: <https://github.com/shapeshifter/shapeshifter-library>

When implementing the protocol, make sure the implementation is compliant to the GOPACS specifications as described in this document. All UFTP traffic must be uniform in order to make sure that every participant is able to communicate with each other.

GOPACS uses (in this documentation) the term Trading Company for every participant that has a contract with the Grid Company. In the UFTP specification the term AGR (aggregator) is used, and in some Capacity Steering Contracts the term CSP is used.

GOPACS environments

Environment	ACC (suggested for testing integrations)	PRD
GOPACS UI	https://app.acc.gopacs.eu	https://app.gopacs.eu
UFTP endpoint This is where you send your messages to	https://clc-message-broker.acc.gopacs-services.eu/shapeshifter/api/v3/message	https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message
Participant API (address book)	https://clc-message-broker.acc.gopacs-services.eu/v2/participants/	https://clc-message-broker.gopacs-services.eu/v2/participants/
API documentation	 Swagger UI	 Swagger UI
Public IPs of GOPACS (for IP whitelisting)	3.75.32.104 3.77.164.86 35.158.231.79	3.121.132.49 3.76.130.111 3.78.82.175
Sign-up link for new CSPs	Sign-up for GOPACS	Sign-up for GOPACS
OAuth2 token endpoint	https://auth.acc.gopacs-services.eu/realms/gopacs/protocol/openid-connect/token	https://auth.gopacs-services.eu/realms/gopacs/protocol/openid-connect/token

Getting started

Summary

To set up for using UFTP messages for flex trading, perform the following steps (not necessarily in this order) which are described below:

1. Account Setup for the GOPACS Application
2. Set up an API client
3. Enter your company details for UFTP and OAuth2.0 on the GOPACS Application
4. Obtaining a bearer token
5. Implement UFTP endpoint
6. Test implementation and settings on the GOPACS Acceptance environment

Account Setup for the GOPACS Application

For New CSPs or Affiliates

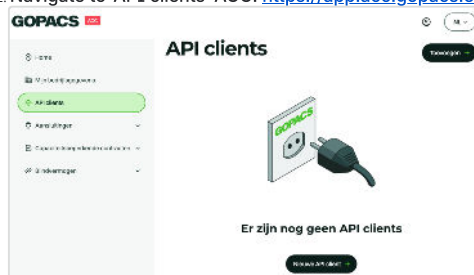
If you are a **new** CSP or affiliate (*aangeslotene*), you can create a company account with user account(s) yourself if you don't already have one. Please note that each environment requires a **separate account**—accounts are not shared across environments.

After completing the sign-up form and accepting the terms, your request will be sent to a grid operator for approval. Once approved, you will receive an email with instructions to finalize your account setup. Only then will you be able to log in to the GOPACS application.

Set up an API client

After signing up to GOPACS, you need to create at least one **API client** to authenticate your company on API requests:

1. Log in to the GOPACS GUI as company admin
2. Navigate to 'API clients' ACC: <https://app.acc.gopacs.eu/api-clients> or PRD: <https://app.gopacs.eu/api-clients>



3. Create a new API client on this page



4. Store the **client ID** and **client secret** somewhere secure (e.g. secrets manager software). Note that the client secret cannot be viewed again.

Enter your company details for UFTP and OAuth2.0 on the GOPACS Application

Please refer to the manual **Company settings for participating in CLC (Capacity Limiting Contracts)** on [Documents and manuals - GOPACS](#)

Obtaining a Bearer token

Use the [OAuth2 client credentials flow](#) to obtain a Bearer token:

1. Configure an OAuth2 client to use the GOPACS OAuth2 token endpoint (see the table above)
2. Configure the client ID and client secret of the API client created via the GOPACS GUI
3. Perform the **OAuth2 client credentials flow**
4. A Bearer token will be returned (access token) which will be valid for a limited time (usually 5 minutes)
5. Include the Bearer token in each HTTP request to the GOPACS API
6. Refresh the Bearer token in time by performing the same flow again

Implement UFTP endpoint

As a Trading Company you need to have an implementation of the UFTP protocol for sending and receiving messages to and from grid operators by API. To receive UFTP messages you need to expose an endpoint compliant to the Shapeshifter UFTP specification, that is ready to accept UFTP messages from GOPACS.

NB. It is also possible to make use of the GOPACS UI to send and receive UFTP Flex messages instead of using an API implementation (Please refer to the manual **Flex trading with Capacity Limiting Contracts using the GOPACS Application**).



Test implementation and settings on the GOPACS Acceptance environment

To make sure that your implementation and settings are correct, please first test on the GOPACS Acceptance environment.

Please refer to the manual **Testing sending and receiving flex messages by the UFTP API** on [Documents and manuals - GOPACS](#)



GOPACS on behalf of a Grid Company or Trading Company

If a Grid Company or Trading Company doesn't have their own UFTP API, they can use the GOPACS platform ([🌐 Homepage - GOPACS](#)) to send and receive messages on their behalf. In this case, the UFTP messages are delivered to a GOPACS endpoint and GOPACS responds on behalf of that participant.

If GOPACS sends a message on behalf of a participant, the SenderDomain is always the UFTP domain of the actual (original) Grid Company or Trading Company, not GOPACS.

Participant API ('Address book')

DNS discovery as described in the UFTP specification is currently not supported by GOPACS.

GOPACS provides an alternative API for discovery of UFTP participant information: the Participant API.

Introduction on the Participant API

There are currently two endpoints for the discovery of UFTP Participant information;

1. Search on EAN and UFTP Role
2. Search on Domain Name and UFTP Role.

When using the information retrieved from the participant API, please know that it does not matter whether the company with which you are communicating on flex trading uses its own UFTP implementation or the GOPACS application. You need to always use the received UFTP participant information for that company for sending the flex message; it will always contain the GOPACS CLC message broker endpoint. The CLC message broker will then determine whether to send the message to the recipients UFTP endpoint or to the GOPACS application.

Search on EAN and UFTP Role

You can retrieve the participant information by searching on the EAN of the contracted grid connection and the UFTP role of the participant (DSO or AGR). As there may be multiple trading companies/aggregators on one grid connection, the response is a list of participants.

Example Participant API request:

```
1 GET /v2/participants/DSO?contractedEan=123456789012345678 HTTP/1.1
2 Accept: application/json
```

Example Participant API response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 [
5   {
6     "domain": "example.com",
7     "publicKey": "VFHpQ4B71gOKrVJAG+HK1zQctr1J3zjk4BYGK79E+c=",
8     "endpoint": "https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message"
9   }
10 ]
```

NB! Since we actively encourage to let all UFTP message go through the CLC Message Broker for reporting purposes, the returned participant 'endpoint' is always the CLC Message Broker endpoint.

Response Status

HTTP status	Possible cause
200	ok (note: if no participants could be found, an empty list is returned with HTTP Status 200)
400	EAN is not 18 characters wide, or <code>uftpRole</code> is not <code>AGR</code> or <code>DSO</code> (<code>CR0</code> is not supported)

Search on Domain Name and UFTP Role

This is typically used to verify and decrypt the payload when receiving a message.

i GOPACS also provides a `v1/participants` endpoint, but this API endpoint has been marked as deprecated, and will be removed in a future release; we *strongly* advise you stop using this endpoint, and instead use the `v2` endpoint instead.

Example Participant API request:

```
1 GET /v2/participants/DSO/example.com HTTP/1.1
2 Accept: application/json
```

Example Participant API response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 {
5   "domain": "example.com",
6   "publicKey": "VFHpQ4B71gOKrVJAG+HK1zQctr1J3zjk4BYGK79E+c=",
7   "endpoint": "https://uftp.example.com/shapeshifter/v3/message"
8 }
```

Response statuses

HTTP status	Possible cause
200	ok
400	<code>uftpRole</code> is not <code>AGR</code> or <code>DSO</code> (<code>CR0</code> is not supported)
404	Participant not found for role + domain name.

ISPs

Certain message types such as `FlexRequest`, `FlexOffer` and `FlexOrder` contain ISPs: Imbalance Settlement Periods.

While the Shapeshifter UFTP specification allows to include not only the ISPs that are requested, but also those that are available (not being steered), GOPACS currently only expects the ISPs that are to be limited under the contract.

ISP Duration

Currently the ISP duration supported is always 15 minutes.

ISP-Duration	Required. Must be <code>PT15M</code>
--------------	--------------------------------------

ISP Start

First ISP of the day is 1. 00:00:00 (inclusive) until 00:15:00 (exclusive)

Second ISP of the day is 2. 00:15:00 (inclusive) until 00:30:00 (exclusive)

Last ISP of the day is 96 or 100 or 92. 23:45:00 (inclusive) until 00:00:00 the next day (exclusive)

ISPs and Daylight Saving time

ISPs with respect to Daylight Saving Time (DST), assuming Europe/Amsterdam and a 15 minute ISP duration:

- On the last Sunday of March when the clock goes from CET (standard) to CEST (summer), the number of ISPs will be 92:
 - ISP 1: 00:00-00:15
 - ISP 2: 00:15-00:30
 - ISP 3: 00:30-00:45
 - ISP 4: 00:45-01:00
 - ISP 5: 01:00-01:15
 - ISP 6: 01:15-01:30
 - ISP 7: 01:30-01:45
 - ISP 8: 01:45-~~03:00~~
 - ISP 9: ~~03:00~~-03:15
 - etc.
 - ISP 92: 23:45-00:00
- On the last Sunday of October when the clock goes from CEST (summer) to CET (standard), the number of ISPs will be 100.
 - ISP 1: 00:00-00:15
 - ISP 2: 00:15-00:30
 - ISP 3: 00:30-00:45
 - ISP 4: 00:45-01:00
 - ISP 5: 01:00-01:15
 - ISP 6: 01:15-01:30
 - ISP 7: 01:30-01:45
 - ISP 8: 01:45-02:00
 - ISP 9: 02:00-02:15
 - ISP 10: 02:15-02:30
 - ISP 11: 02:30-02:45
 - ISP 12: 02:45-~~02:00~~
 - ISP 13: ~~02:00~~-02:15
 - etc.
 - ISP 100: 23:45-00:00
- On any other day, the number of ISPs will be 96.

ISP (Min, Max) Power

 Important notice: This is the absolute capacity in watts, not the deviation as stated in the Shapeshifter specification. This might change when V4.x of the Shapeshifter specification is implemented.

- The GUI shows the power values in MW
- In the UFTP messages, power values are in W

Depending on the capacity steering direction:

- When limiting feed-in:
 - `MaxPower` = 0
 - `MinPower` is the maximum allowed feed in as a negative number. E.g. when limiting the feed in to 3 MW, `MinPower` = -3000000. If no feed in is allowed at all, `MinPower` = 0
 - `MinPower` must be a multiple of -1000W (in other words, the limitation is specified in steps of 1kW) and must also match the steps defined in the contract.
- When limiting offtake:
 - `MinPower` = 0
 - `MaxPower` is a positive number specifying the maximum allowed power offtake in Watts.
 - `MaxPower` must be a multiple of 1000W.
- When **deploying feed-in**:
 - `MaxPower` < 0, the minimum required feed-in as a negative number.
 - `MinPower` < `MaxPower`, the maximum allowed feed-in as a negative number.
- When **deploying off-take**:
 - `MinPower` > 0, the minimum required off-take as a positive number.
 - `MaxPower` >= `MinPower`, the maximum allowed off-take as a positive number.

We don't support the following cases currently:

- Both `MinPower` < 0 and `MaxPower` > 0.
- Both `MinPower` = 0 and `MaxPower` = 0

Validations of (Min,Max) Power against Contract

Power must meet the steps for activation relative to the minimum/maximum flex capacity on the Contract applicable to the given ISP.

E.g. minimum flex capacity = 35000 W, steps for activation = 10000 W, then ISP power 45000 W would be valid, but 50000 W would be invalid.

Baseline (proposal, not supported yet)

i This is a proposal and is not supported yet.

ISP.DefaultBaseline	Capacity in the default situation that would occur if no flexibility were activated. This is usually equal to the contracted transport capacity (GTV).
Not supported yet	Example: the GTV is 100 MW. No flexibility is activated yet: the DefaultBaseline is 100 MW. If 25 MW of flexibility was already activated: the DefaultBaseline is still 100 MW. This is a proposal and is not supported yet.
ISP.Baseline	Capacity baseline before this flexibility was requested. If flex capacity was activated earlier, in a different conversation, then this is the capacity with the deviation applied.
Not supported yet	Example: the DefaultBaseline is 100 MW. An earlier conversation has activated 25 MW of flexibility. When this FlexRequest is sent to request the activation of another 20 MW of flexibility, the Baseline would be 75 MW and the MaxPower would be 55 MW. This is a proposal and is not supported yet.

SignedMessage

Example of a SignedMessage HTTP request (some headers omitted for clarity):

1	POST /shapeshifter/api/v3/message HTTP/1.1
2	Accept: application/json (or */* or omit)
3	Authorization: Bearer ...
4	Content-Type: text/xml
5	
6	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
7	<SignedMessage
8	SenderDomain="dso.nl"
9	SenderRole="DSO"
10	Body="...">

Attribute	GOPACS expectation
SenderDomain	Grid Company must log in to the GOPACS UI and configure UFTP domain, public key and endpoint in GOPACS, prior to receiving or sending messages. When GOPACS is sending messages on behalf of Grid Company or Trading Company, then the SenderDomain is equal to the domain of actual UFTP participant.
SenderRole	Must be DSO for a Grid Company. Must be AGR for a Trading Company. Other values are not supported. GOPACS is not considered a 'party' in the UFTP exchange and therefore does not have a 'role'. When GOPACS is sending messages on behalf of Grid Company or Trading Company, then the SenderRole is equal to the role of actual UFTP participant.
Body	Base64 encoded payload that will be decrypted using <code>crypto_sign_open</code> (see Public-key signatures Libsodium documentation). GOPACS does not support sealing and unsealing of messages (just signing). GOPACS uses the SenderDomain and SenderRole to lookup the public key that is used for verifying the signature of the message. Encryption-in-transit is covered by enforcing TLS connections. Encryption-at-rest should be done using common security practices and tools by the implementing system. GOPACS uses Lazysodium which is a Java wrapper over the Libsodium library.

Response statuses

HTTP status	Possible cause
200	Message has correct signature, was XSD valid and will be processed asynchronously.
400	Wrong content type Technical XSD validation error (no functional validations yet) Error during XML deserialization
401	Bearer token not provided or invalid. Public key of sender not found or incorrect. Message signature could not be verified with public key of sender.
403	Bearer token not authorized to perform this request.



5xx	Unexpected (temporary) error on GOPACS side.
-----	--

Message types with examples and field specifications

FlexRequest

Sent by the Grid Company to the Trading Company.

Example of a FlexRequest message for a Capacity Steering Contract, where the contracted capacity is 100 MW, the requested limit is 50 MW:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexRequest
3   Version="3.0.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="agr.nl"
6   TimeStamp="2021-10-29T06:54:26.861Z"
7   MessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
8   ConversationID="48cd3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   ServiceType="CBC"
10  ISP-Duration="PT15M"
11  TimeZone="Europe/Amsterdam"
12  Period="2021-10-30"
13  ContractID="A-AA-A-12345"
14  CongestionPoint="eam.265987182507322951"
15  Revision="1"
16  ExpirationDateTime="2021-10-29T22:15:00.0000Z">
17    <ISP Start="48" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500000000"/>
18    <ISP Start="49" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500000000"/>
19    <ISP Start="50" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500000000"/>
20    <ISP Start="51" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500000000"/>
21  </FlexRequest>

```

Example of the same FlexRequest message signed with the ACC key:

[illegible]

All the validations from the UFTP specification apply. On top of that GOPACS has some additional restrictions with respect to the usage of UFTP message for CSC:

Attribute/Element	GOPACS additional restrictions				
Version	Must be 3.0.0 (currently).				
SenderDomain	To enable receiving or sending messages, the Grid Company must log in to the GOPACS UI and configure with its company details their UFTP domain, public key and endpoint.				
RecipientDomain	To enable receiving or sending messages, the Trading Company must log in to the GOPACS UI and configure with its company details their UFTP domain, public key and endpoint.				
TimeStamp	Parsing supports different offsets and handles accordingly. The offset can be either in "+HH:mm:ss" format or "Z". GOPACS ignores the milliseconds part when parsing. GOPACS always sends either a UTC timestamp (no offset and "Z" suffix) or a timestamp in the Europe/Amsterdam timezone (offset +01:00 or +02:00 depending on DST). The milliseconds part can be between 0 and 9 digits where the omitted digits are implied to be zero.				
Revision	Required. Must be 1. Revisions are currently <u>not</u> supported yet.				
ServiceType	Optional. The ServiceType specifies which type of flexibility is being requested. Note that ServiceType is still CBC instead of the new naming CSC. <table border="1"> <thead> <tr> <th colspan="2">ServiceType</th></tr> </thead> <tbody> <tr> <td>CBC</td><td>Capacity Limiting Contract (<i>Capaciteitsbeperkingscontract</i>)</td></tr> </tbody> </table>	ServiceType		CBC	Capacity Limiting Contract (<i>Capaciteitsbeperkingscontract</i>)
ServiceType					
CBC	Capacity Limiting Contract (<i>Capaciteitsbeperkingscontract</i>)				
ISP-Duration	Required. See ISPs .				
TimeZone	Required. Must be Europe/Amsterdam				

Period	Required.  The day of congestion. Format: YYYY-MM-DD always interpreted in Europe/Amsterdam time zone. <u>Any offset is ignored.</u> The message must be sent before 12:00:00 the day before Period. Examples: • If the message is received before 12:00:00, then the Period may be tomorrow or later. • If the message is received after 12:00:00, then the Period must be the day after tomorrow or later.
ExpirationDateTime	 The expiration date time must be no later than 12:00:00 the day before the day of congestion (Period).
ContractID	Functional required. Typical format: A-AA-A-12345 Trading Company must log in to the GOPACS UI and register the CLC contract, prior to receiving or sending messages.
CongestionPoint	ean. [0-9]{18} Must be a known EAN of a preregistered CLC contract in GOPACS. Does <u>not</u> have to be known as Grid Connection in GOPACS.
ISP	Required. See ISPs.
ISP.Start	Required. See ISPs.
ISP.Duration	
ISP.Disposition	Must be Requested See ISPs.
ISP.MinPower	See ISPs.
ISP.MaxPower	

FlexRequestResponse

Example FlexRequestResponse:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexRequestResponse
3   SenderDomain="agr.nl"
4   RecipientDomain="dso.nl"
5   TimeStamp="2021-10-29T06:54:36.443796Z"
6   MessageId="7f04a6b-8f42-4b92-911e-b26ff85525067"
7   ConversationId="448cd3d2-56c0-436c-8d5a-6ff6c3dc358d"
8   Result="Accepted"
9   FlexRequestMessageId="d3ae4836-55b1-4084-b54e-34107b22648c"/>

```

Example of the same FlexRequestResponse message signed with the ACC key:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <SignedMessage SenderName="agril" SenderRole="AGR">
3   Body="3fD4f545k6h7KtQ6s4Jd0XCpA1Lzrx1yaej4db0Sufsc7Z60ZsE4UfVM3bhy0+3
4   V9WdE7oR5LdN4TgXBGV4UwVsdV4WtG3BvbnNIDQoGAgCAGfZlcmNhp249lJmUcM4wI6kG
5   AgCAGfCAGIBTzW5XZk5Zj6h4wF9mG4vpcSINCiAGfCAGfCAGUwYpBzW50R9GwYlupPSkE
6   2Buhm9dUqQoGAgCAGfFRpbWtG7dGfCfD0iMJ5Am5X0cyM0vQwNjo1Nd0zN40NDM30YUwYlWlNCI
7   AgCAGfCAGTWvzeZn2UEH3P5J3BMnGUG20CfImNqL7RtOTRiOTeZsS1iMzJmO0DUiMjUwNkYzLz
8   gIAGfCAGfENbvznFmZndH6GvkEjPSj0nKMi01NmMwL7QzNmMwGQYl1S02ZjYwNkYzLz
9   OQ0GfCAGfCAGfJf3f3v4D0iUqWYXZB0Z0QdQoGAgCAGfEzS2XhS2XF1XN0T7wvzeZnZ
10  UEDPjRkM2FINDgZ01NwLk1QWQdQYlU50ZNdNmWzN2lYmY0OGMLdC=">

```

Or when it is rejected:

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <FlexRequestResponse
3    Version="3.0.0"
4    SenderDomain="aggregator.org"
5    RecipientDomain="ufip.dso.nl"
6    TimeStamp="2021-10-29T06:54:36-0437962Z"
7    MessageID="7f04e6b-f842-4b92-911e-2db6f8552067"
8    ConversationalID="48cdc3d2-56c0-436c-8d5a-6ffc3dc358d"
9    Result="Rejected"
10   RejectionReason="Reference Problem mismatch"
11   FlexRequestMessageID="d3ae48356-5b1-4084-b54e-34107b22648c"/>

```

Example of the same FlexRequestResponse message signed with the ACC key:

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <SignedMessage SenderDomain="aggregator.org" OrderCode="AGR"
3      Body="m+H5FFII0EpuMlL+VpY6KkHb8OtrCtU/VhLyNhIaWzCEh743SPeLrSP9pbWtJdmuSfD
4      S3Yl3amQ4uGVCzGxbG4U4mVsdVzdzfJl3CBvbnNDmKlCagIcAGlFzIcnNbp249jM1McAw4lgKlC
5      AgICaCBTZW7X5JkE12b4A9lFcmZ9J3l2f3CB33n3lmgKlCAGlCAGlCBSZWNpCGLlnbRbE2+
6      aW49lNvdInAuZHnVn5slgKlCAGlCAGlCBZJUdV1U13RbX5A9lJjM+EmtAtAmJlMUdY6NTG6mZyUnd
7      QzNkZ2Ml0idQAGlCAGlE1lC3hZhZ3BD0iN2+Vz3RlRlGzjGj0M0Y0jklTlxMwUyIzZjZg1
8      NT1lMlDy3lgKlCAGlCAGlCBZ52Z2ZJYzXrpb25ZDmNDhJzGzGmZDlntZjM00mZzJlTlxNWEInE
9      YZ2YmZGm1MzhlgKlCAGlCAGlCBSZXNlJlH09lJlIamVJdVlgKlGKlCAGlCAGlCBSZWpYl3P3rb25
10     WZFbZ49lJlJmZvJw5Zj5ZG3Ujb2GqWlzbWfY02lGQAGlCAGlEAGlEZhS5ZKZFNZ0UWVwrcZ
11     FxN7X0M2F2MlFN2GdN1NwLxLTQwQDQYlU02S0ENdEwNylY0OGMlZ4="?>

```

FlexOffer

Sent by the Trading Company to the Grid Company as answer to a FlexRequest.

Note: The FlexOffer message type does not have the field `serviceType` as FlexRequest and FlexOffer do. This is because it is not needed on this message type.

Example of a FlexOffer message for a CSC contract:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
2 <FlexOffer
3 Version="3.0.0"
4 SenderDomain="agr.nl"
5 RecipientDomain="dso.nl"
6 Timestamp="2021-10-29T06:54:36.8868538Z"
7 MessageID="338ed243-5517-4400-962e-2b7b812c468c"
8 ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9 ISP-Duration="PT15M"
10 TimeZone="Europe/Amsterdam"
11 Period="2021-10-30"
12 CongestionPoint="ean.265987182507322951"
13 ExpirationDateTime="2021-10-29T10:30:00Z"
14 FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
15 ContractID="A-AA-A-12345"
16 BaselineReference=""
17 Currency="EUR"
18 <OfferOption OptionReference="ba40a5f8-849b-4fe6-958f-e628a1653558"
19 Price="0.00"
20 <ISP Start="58" Duration="1" Power="50000000"/>
21 <ISP Start="59" Duration="1" Power="50000000"/>
22 <ISP Start="60" Duration="1" Power="50000000"/>
23 <ISP Start="61" Duration="1" Power="50000000"/>
24 </OfferOption>
25 </FlexOffer>
```

Example of the same FlexOffer message signed with the ACC key:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <SignedMessage SenderDomain="agr.nl" SenderRole="AGR"
3 Body="whfSzCmP9OmI2y/6lfz/7a6hRvz7Am5waAwmtKT52kbEG90hqHTj4DisCRNnmGHmDjAsq6AyNB/YG+gDnZ5BjxGbGV4T2ZmZXINCiAgICAgICAgVmVyc2lvbj0iMy4wLjAiDQogICAgICAgIFNlbmRl"
```

⚠ Unsolicited FlexOffer messages are rejected by GOPACS. There must always be a preceding FlexRequest.

⚠ At most 1 FlexOffer message may be sent as part of a conversation. All successive FlexOffer messages will be rejected by GOPACS (for now).

Attribute/Element	
D-PrognosisMessageID	Ignored.
BaselineReference	Ignored.
Currency	Must be EUR .
Price	Must be 0.00 (for now). Must comply with ISO 4217 0.00 , 0.0 and 0 are all allowed and considered equal in the GOPACS implementation.
OfferOption	⚠ Exactly 1 OfferOption element is expected.
OfferOption.MinActivationFactor	Optional. Ignored.
ISP.Power	Depending on the contractual agreements between AGR and DSO the Power either is equal to what has been requested in the FlexRequest or it can deviate. Limitation: In the case of a limitation on production, Power refers to the MinPower attribute of the ISP . In case the consumption is limited, Power refers to the MaxPower attribute. Deployment: In case of the a deployment on production, Power refers to the MaxPower attribute of the ISP. In case the consumption is deployed, Power refers to the MinPower attribute. See ISPs . It is allowed to send an offer on a subset of the requested ISPs.

Other attributes like Period, CongestionPoint, ContractID, etc. must be equal to the original FlexRequest.

FlexOfferResponse

Example FlexOfferResponse:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOfferResponse
3 Version="3.0.0"
4 SenderDomain="dso.nl"
5 RecipientDomain="agr.nl"
6 Timestamp="2021-10-29T06:54:36.4437962Z"
7 MessageID = UUID
8 ConversationID = UUID
9 FlexOfferMessageID = UUID
10 Result = ("Accepted" | "Rejected")
11 RejectionReason = String (Only if Result = "Rejected")
12 >
```

FlexOrder (for CSC)

Example of a FlexOrder message for a CSC contract, where the off-take transport capacity is limited to 50 MW (the original contracted transport capacity is 100 MW):

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrder
3 Version="3.0.0"
4 SenderDomain="dso.nl"
5 RecipientDomain="agr.nl"
6 Timestamp="2021-10-29T06:55:36.518Z"
7 MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
8 ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9 ISP-Duration="PT15M"
10 TimeZone="Europe/Amsterdam"
```

```
11 Period="2021-10-30"
12 CongestionPoint="ean.265987182507322951"
13 FlexOfferMessageID="338ed243-5517-4400-962e-2b7b812c468c"
14 ContractID="A-AA-A-12345"
15 Price="0.00"
16 Currency="EUR"
17 OrderReference="None">
18 <ISP Start="58" Duration="1" Power="50000000"/>
19 <ISP Start="59" Duration="1" Power="50000000"/>
20 <ISP Start="60" Duration="1" Power="50000000"/>
21 <ISP Start="61" Duration="1" Power="50000000"/>
22 </FlexOrder>
```

Attribute/Element	
ServiceType	Optional for CSC. Either <code>CBC</code> or leave empty
Currency	Must be <code>EUR</code> .
Price	Must equal the Price in the FlexOffer.
OrderReference	May be filled by the calling grid company for settlement process. If the grid operator is using GOPACS for UFTP, and they start a request from the GUI, this field is filled with a generated UUID.
ISP	 Currently GOPACS orders exactly what was offered (if on behalf of a Grid Company) - including ISPs and min activation factor.
ISP.Power	Depending on the contractual agreements between AGR and DSO the <code>Power</code> either is equal to what has been requested in the <code>FlexRequest</code> or it can deviate. In the case of a limitation on production, power refers to the <code>MinPower</code> attribute of the <code>FlexRequest</code> . In case the consumption is limited, power refers to the <code>MaxPower</code> attribute. See ISPs .

FlexOrder (for TDTR and VVTR)

This requires version 3.1.0.

Example of a FlexOrder message for a TDTR contract, where the transport capacity is limited to 50 MW (the original contracted transport capacity is 70 MW):

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrder
3   Version="3.1.0"
4   SenderDomain="dso.nl"
5   RecipientDomain="agr.nl"
6   Timestamp="2021-10-29T06:55:36.518Z"
7   MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
8   ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
9   ServiceType="TDTR"
10  ISP-Duration="PT15M"
11  TimeZone="Europe/Amsterdam"
12  Period="2021-10-30"
13  CongestionPoint="ean.265987182507322951"
14  ContractID="00000001"
15  Price="0.00"
16  Currency="EUR"
17  OrderReference="None">
18 <ISP Start="58" Duration="1" Power="50000000"/>
19 <ISP Start="59" Duration="1" Power="50000000"/>
20 <ISP Start="60" Duration="1" Power="50000000"/>
21 <ISP Start="61" Duration="1" Power="50000000"/>
22 </FlexOrder>
```

Attribute/Element		
Version	Must be 3.1.0 for ATR.	
ServiceType	Type of ATR contract.	
	ServiceType	
	TDTR	Time-bounded transport right (tijdsduurgebonden transportright)
	VVTR	Non firm transport right (Volledig Variabel Transportrecht)
ContractID	Unique number of the ATR contract.	
Currency	Must be EUR .	
Price	Must be 0.00 . But is ignored for now.	
OrderReference	May be filled by the calling grid company for settlement process.	
ISP	ISPs that are to be limited under the contract. See ISPs of FlexRequest .	
ISP.Power	See ISPs.	

FlexOrderResponse

i An “Accepted” response from the Trading Company means that there is a binding agreement.

Example FlexOrderResponse:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <FlexOrderResponse
3   Version      = "3.0.0"
4   SenderDomain = "dso.nl"
5   RecipientDomain = "arg.nl"
6   Timestamp    = "2021-10-29T06:54:36.4437962Z"
7   MessageID    = UUID
8   ConversationID = UUID
9   FlexOrderMessageID = UUID
10  Result       = ("Accepted" | "Rejected")
11  RejectionReason = String (Only if Result = "Rejected")
12 />
```

CSC with “deployment” instead of “limitation” (under construction, not ready yet)

i Functionality is under construction, not fully ready to be used yet.

Introduction

Next to flex trading for capacity limitation (limiting feed-in or off-take, NL: beperking), we are working on adding flex trading for capacity deployment (forcing feed-in or off-take, NL: inzet).

Contracts

Contracts must be registered in GOPACS before calling them. The ContractID is the identifying attribute e.g. T-CA-I-12345.

Note: If a connection has both a Feed-In-congestion contract (invoedingscongestiecontract) and a Off-take-congestion contract (afnamecongestiecontract), then both contracts must have a unique ContractID and be registered separately.

MVP: leave fees empty (or 0) as currently only 'limitation' fees can be registered.

Differences in contract registration compared to CLC (Capacity Limiting Contracts / Capaciteitsbeperkendcontract):

- The two contract types **Production** and **Consumption** are changed to two new types **Feed in congestion** and **Off take congestion**. Existing contracts will be migrated to these two new types.
- When activating (calling) these new contract types:
 - Activating a **Feed in congestion** contract can result in two solutions: **Feed in limitation** and **Off take deployment**
 - Activating a **Off take congestion** contract can result in two solutions: **Off take limitation** and **Feed in deployment**
- Some terminology in the contract data model changes to adjust to the new options with capacity steering direction deployment.

UFTP messages

UFTP message follow the same message flow for CSC as compared to the previous CLC.

Activating a **Off take congestion** contract for **Feed in deployment**:

- FlexRequest
 - ServiceType: CSC? tbd MVP: allowed to be left empty
 - Period
 - ExpirationDateTime: tbd
 - ISP:
 - Disposition: REQUESTED
 - MinPower: Positive value, the “at least” limit, may be 0 to indicate there is no must run expected
 - MaxPower: Positive value, usually the original contracted transport capacity
- FlexRequestResponse
- FlexOffer:
 - ISP:
 - Power: Value between MinPower and MaxPower from the FlexRequest. This means the plant CAN take at least Power from the grid.
- FlexOfferResponse
- FlexOrder:
 - ServiceType: same as in FlexRequest, only if AGR supports 3.1.0 or later
 - ISP:
 - Power: Power from FlexOffer. This means the plant MUST take Power from the grid.
- FlexOrderResponse: agreement is reached

Activating a **Feed in congestion** contract for **Off take deployment**:

- FlexRequest, same as above, except:
 - ISP:
 - MinPower: Negative value, usually the original contracted transport capacity
 - MaxPower: Negative value, the “at least” limit, may be 0 to indicate there is no must run expected
- FlexRequestResponse, same as above

- FlexOffer:
 - Power: Value between MinPower and MaxPower from the FlexRequest. This means the plant CAN feed at least Power into the grid.
- FlexOfferResponse, same as above
- FlexOrder:
 - ServiceType: same as in FlexRequest, only if AGR supports 3.1.0 or later
 - ISP:
 - Power: = Power from FlexOffer. This means the plant MUST feed Power into the grid.
- FlexOrderResponse: same as above

Validations

Because steering needs more variables and has more considerations as compared to limitation, we need to validate more constraints when calling: e.g. time between calls, minimum length.

🔍 Is GOPACS going to be the 'gate keeper'? If so, we need to have a technical way of validating and also reporting validation errors back to the grid operator. Currently Shapeshifter may or may not support this correctly. Also it brings more responsibility to GOPACS: can and will GOPACS do this. Any validations need to be aligned across grid operators. Any changes need to be coordinated. GOPACS could provide all the necessary data for validation, but leave the responsibility at the grid operator.

Testing receiving and sending flex messages

Acceptance environment

On the acceptance environment only, we have created a testing option to test whether your UFTP implementation can receive and send flex messages. With your trading company account you can trigger a flex request, which will be sent to your endpoint. Please refer to the manual **Testing receiving and sending flex messages by UFTP API** on [Documents and manuals - GOPACS](#).

Production environment

After you have completed your testing on the Acceptance environment successfully, you might want to check your environment specific UFTP settings for the Production environment. The UFTP protocol provides the `TestMessage` and `TestMessageResponse` message types for testing purposes. Both are Supported by the clc-message-broker. They can be used to test your UFTP settings on the Production environment

Testing sending messages and receiving a response message

If a `TestMessage` is sent to the clc-message-broker with a recipient that is using GOPACS for UFTP, a `TestMessageResponse` will be sent back automatically.

Test receiving messages and responding with a response message

To test receiving a `TestMessage` and responding with a `TestMessageResponse` you can ask your grid company to send a `TestMessage` with your uftp implementation as recipient. The clc-message-broker will then forward this test message to your uftp implementation.

GOPACS implementation specifics

- ✖ FlexOfferRevocation not supported yet
- ✖ FlexRequest revisions not supported yet
- ✖ Other message types are not supported yet

⚠ A duplicate MessageID is immediately responded to with a 400 Bad Request and **not** a 200 OK followed by "Rejected" response as described in the specification!

⚠ MinPower, MaxPower and Power are implemented as absolute values.

ℹ After a 200 OK is returned, a received message is immediately processed by GOPACS. An accepted or rejected response is sent back almost instantaneously.

ℹ The user receives realtime email notifications when a FlexRequest, FlexOffer or FlexOrder is received, rejected or failed to deliver.

ℹ An outgoing UFTP message is retried every 3 minutes for a maximum of 5 tries. After that, the user and GOPACS DevOps team are notified of a failure to deliver. Specifically on a 400 Bad Request, a message is not retried.

ℹ Typically there will be at most 15 mins between FlexRequest and FlexOrder.