UFTP messages for flex trading with Capacity Limit Contracts at GOPACS

Last updated: 19-02-2025

GOPACS supports flex trading with Capacity Limit Contracts (*capaciteitsbeperkende contracten* or *lange termijncontracten*) through the exchange of Shapeshifter UFTP messages.

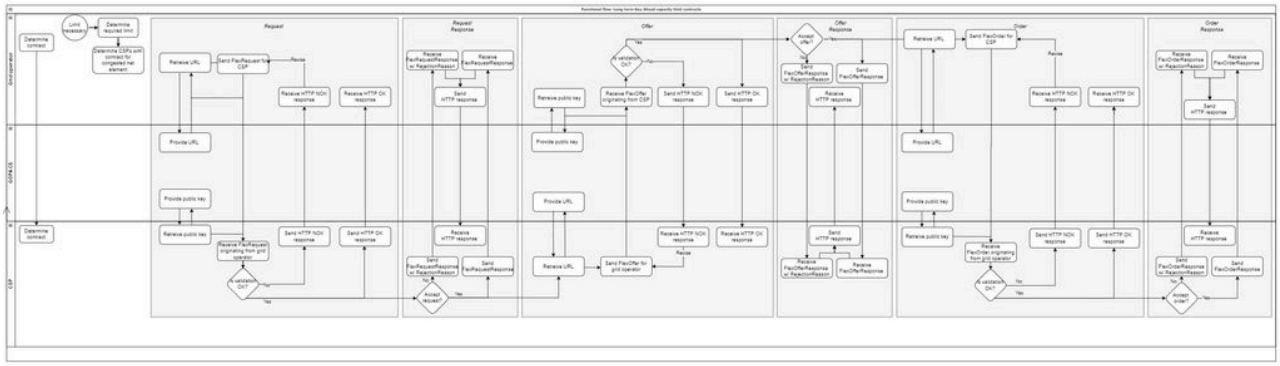
Table of contents

Usage of the UFTP protocol for CLC

Overview of the GOPACS and UFTP message exchange:

The Trading Company needs to have an implementation of the UFTP protocol for sending and receiving messages to and from grid operators. The implementation must be compliant to the UFTP-specification: https://github.com/shapeshifter/shapeshifter-specification. Currently GOPACS supports version 3.0.0 of the specification with some restrictions which are described separately in this document.

The part where the URL and the public key is retrieved, is GOPACS/CLC specific.



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

NB. It is also possible to make use of the GOPACS UI to send and receive UFTP Flex messages.

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.

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 CLC 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 Limit 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

UFTP Public Key This is the public key that can be used to verify messages sent from GOPACS UI on behalf of a DSO/CSP not having their own implementation.	`VFHpQ4B71gOKrVJAG+HK1zQctr1J3zjkK4BYGK79E+c=`	`pI/s07d9fvoatyo5gVLZKLfdqQmdkaMAIgConmcl76U=`
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

Have an account for the GOPACS application

If you are a **new CSP** you can signup for a new account yourself if you don't have it it. Each environment requires you to signup for a separate account; accounts are not shared across environments.

After you complete the sign-up form and accept the terms, a grid operator will receive a notification and has to approve your request before you will receive the email to finish your account and 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'
3. Create a new API client
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

Add your company details for UFTP communication and OAuth2.0 on the "My company details" / "Mijn bedrijfsgegevens" page, on the "CLC" / "CBC" tab.

UFTP details

- UFTP domain (mandatory)
- UFTP endpoint (mandatory, must be https)

- UFTP public key (mandatory)

These details will be your “Participant information” that other companies will retrieve from the GOPACS Participant endpoint to send UFTP requests to your company (see next paragraph “Participant API”)

OAuth2.0

UFTP requests to the GOPACS endpoint must include an OAuth2 Bearer token in the Authorization header of each HTTP request. See next paragraph on “Obtainint a Bearer token”

For receiving UFTP requests, using OAuth2.0 is optional. After selecting the optional checkbox to use OAuth2.0 for receiving UFTP requests, the following fields appear:

- Token URL (mandatory)
- Scope (optional, space separated list - scopes that your own authentication provider requires)
- Client Id (mandatory - from set up API client in the previous paragraph)
- Client secret (mandatory - from set up API client in the previous paragraph)

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

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.

There are currently two endpoints for the discovery of UFTP Participant information; 1) Search on EAN and UFTP Role and 2) Search on Domain Name and UFTP Role

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:

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

Example Participant API response:

```
HTTP/1.1 200 OK
Content-Type: application/json

```

```
[
  {
    "domain": "example.com",
    "publicKey": "VFHpQ4B71gOKrVJAG+HK1zQctr1J3zjkK4BYGK79E+c=",
    "endpoint": "https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message"
  }
]
```

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 `uftpRole` is not `AGR` or `DSO` (`CRO` is not supported)

Search on Domain Name and UFTP Role

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

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:

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

Example Participant API response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "domain": "example.com",
  "publicKey": "VFHpQ4B71gOKrVJAG+HK1zQctr1J3zjkK4BYGK79E+c=",
  "endpoint": "https://uftp.example.com/shapeshifter/v3/message"
}
```

Response statuses

HTTP status	Possible cause
200	ok
400	`uftpRole` is not `AGR` or `DSO` (`CRO` is not supported)
404	Participant not found for role + domain name.

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 (https://gopacs.eu/) 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.

SignedMessage

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

```
POST /shapeshifter/api/v3/message HTTP/1.1
Authorization: Bearer ...
Content-Type: text/xml

<SignedMessage
  SenderDomain="dso.nl"
  SenderRole="DSO"
  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 `crypto_sign_open` (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 FlexRequest:

```
<FlexRequest
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="agr.nl"
  TimeStamp="2021-10-29T06:54:26.861Z"
  MessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-30"
  ContractID="A-AA-A-12345"
  CongestionPoint="ean.265987182507322951"
  Revision="1"
  ExpirationDateTime="2021-10-29T22:15:00.0000Z">
  <ISP Disposition="Requested" MinPower="0" MaxPower="50000000" Start="48" Duration="1"/>
  <ISP Disposition="Requested" MinPower="0" MaxPower="50000000" Start="49" Duration="1"/>
  <ISP Disposition="Requested" MinPower="0" MaxPower="50000000" Start="50" Duration="1"/>
  <ISP Disposition="Requested" MinPower="0" MaxPower="50000000" Start="51" Duration="1"/>
</FlexRequest>
```

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

```
 <SignedMessage SenderDomain="dso.nl" SenderRole="DSO"
               Body="SwGOfSa4bZ9ghmuQmm7lvTkgDvF2F/dy1A3qqe7qkciiH/qyIuXdAAxfV8+jqW8Gc91pcqMoYr8i
FUeDmIvJDjxGbGV4UmVxdWVzdA0KICAgICAgICBWZXJzaW9uPSIzLjAuMCINCiAgICAgICAgU2VuZGVyRG
9tYWluPSJkc28ubmwiDQogICAgICAgIFJlY2lwaWVudERvbWFpbj0iYWdyLm5sIg0KICAgICAgICBUaW1l
U3RhbXA9IjIwMjEtMTAtMjlUMDY6NTQ6MjYuODYxWiINCiAgICAgICAgTWVzc2FnZUlEPSJkM2FlNDgzNi
01NWIxLTQwODQtYjU0ZS0zNDEwN2IyMjY0OGMiDQogICAgICAgIENvbnZlcnNhdGlvbklEPSI0OGNkYzNk
Mi01NmMwLTQzNmMtOGQ1YS02ZjZjYzNkYzUzOGQiDQogICAgICAgIElUC1EdXJhdGlvbj0iUFQxNU0iDQ
ogICAgICAgIFRpbWVab25lPSJFdXJvcGUvQW1zdGVyZGFtIg0KICAgICAgICBQZXJpb2Q9IjIwMjEtMTAt
MzAiDQogICAgICAgIENvbnRyYWN0SUQ9IkEtQUEtQS0xMjM0NSINCiAgICAgICAgQ29uZ2VzdGlvblBvaW
50PSJlYW4uMjY1OTg3MTgyNTA3MzIyOTUxIg0KICAgICAgICBSZXZpc2lvbj0iMSINCiAgICAgICAgRXhw
               aXJhdGlvbkRhdGVUaW1lPSIyMDIxLTEwLTI5VDIyOjE1OjAwLjAwMDBaIj4NCiAgICA8SVNQIERpc3Bvc2
               l0aW9uPSJSZXF1ZXN0ZWQiIE1pblBvd2VyPSIwIiBNYXhQb3dlcj0iNTAwMDAwMDAiIFN0YXJ0PSI0OCIg
               RHVyYXRpb249IjEiLz4NCiAgICA8SVNQIERpc3Bvc2l0aW9uPSJSZXF1ZXN0ZWQiIE1pblBvd2VyPSIwIi
               BNYXhQb3dlcj0iNTAwMDAwMDAiIFN0YXJ0PSI0OSIgRHVyYXRpb249IjEiLz4NCiAgICA8SVNQIERpc3Bv
               c2l0aW9uPSJSZXF1ZXN0ZWQiIE1pblBvd2VyPSIwIiBNYXhQb3dlcj0iNTAwMDAwMDAiIFN0YXJ0PSI1MC
               IgRHVyYXRpb249IjEiLz4NCiAgICA8SVNQIERpc3Bvc2l0aW9uPSJSZXF1ZXN0ZWQiIE1pblBvd2VyPSIw
```

```
17                IiBNYXhQb3dlcj0iNTAwMDAwMDAiIFN0YXJ0PSI1MSIgRHVyYXRpb249IjEiLz4NCjwvRmxleFJlcXVlc3Q+"/>
```

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 CLC:

Attribute/Element	**GOPACS additional restrictions**
Version	Must be `3.0.0` (currently).
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.
RecipientDomain	Trading Company must log in to the GOPACS UI and configure UFTP domain, public key and endpoint in GOPACS, prior to receiving or sending messages.
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 not supported yet.
ISP-Duration	Required. Must be `PT15M`
TimeZone	Required. Must be `Europe/Amsterdam`
Period	Required. ⚠ The day of congestion. Format: `YYYY-MM-DD` always interpreted in `Europe/Amsterdam` time zone. Any offset is ignored. 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 not have to be known as Grid Connection in GOPACS.
ServiceType	Optional. Ignored.
ISP	⚠ GOPACS currently only expects the ISPs that are to be limited under the contract.
ISP.Start ISP.Duration	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 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.Disposition	Must be `Requested` GOPACS expects only the ISPs that are to be limited.
ISP.MinPower ISP.MaxPower	⚠ Expects power in **Watts** as per the Shapeshifter specification 3.0.0. • The GUI shows the power values in MW • Via de API, power values are in W Depending on the capacity limiting direction: • One of `MinPower` or `MaxPower` will always be 0: ◦ When limiting **production**, `maxPower` = 0 ◦ When limiting **consumption**, `minPower` = 0 • The other is 0 or a multiple of 1000 W (limitation is done in steps of 1000 W) ◦ When limiting **production**, `minPower` = 0 or `minPower` <= -1000 (W) ◦ When limiting **consumption**, `maxPower` = 0 or `minPower` >= 1000 (W)

FlexRequestResponse

Example FlexRequestResponse:

```
<FlexRequestResponse
```

```
  Version="3.0.0"
  SenderDomain="agr.nl"
  RecipientDomain="dso.nl"
  TimeStamp="2021-10-29T06:54:36.4437962Z"
  MessageID="7f0f4e68-f842-4b92-911e-b26f85525067"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  Result="Accepted"
  FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"/>
```

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

```
<SignedMessage SenderDomain="agr.nl" SenderRole="AGR"
               Body="3fD4Ie5Jk6h7k6TQaEsJ4Vego1CXpA/1ztx1yaej1db0SnufscFZZy630EsUJFVM3Ihy0+3
               V9WDcWOR5LdnJATxGbGV4UmVxdWVzdFJlc3BvbnNlDQogICAgICAgIFZlcnNpb249IjMuMC4wIg0K
               ICAgICAgICBTZW5kZXJEb21haW49ImFnci5ubCINCiAgICAgICAgUmVjaXBpZW50RG9tYWluPSJkc
               28ubmwiDQogICAgICAgIFRpbWVTdGFtcD0iMjAyMS0xMC0yOVQwNjo1NDozNi40NDM3OTYyWiINCi
               AgICAgICAgTWVzc2FnZUlEPSI3ZjBmNGU2OC1mODQyLTRiOTItOTExZS1iMjZmODU1MjUwNjciDQo
               gICAgICAgIENvbnZlcnNhdGlvbklEPSI0OGNkYzNkMi01NmMwLTQzNmMtOGQ1YS02ZjZjYzNkYzUz
               OGQiDQogICAgICAgIFJlc3VsdD0iQWNjZXB0ZWQiDQogICAgICAgIEZsZXhSZXF1ZXN0TWVzc2FnZ
               UlEPSJkM2FlNDgzNi01NWIxLTQwODQtYjU0ZS0zNDEwN2IyMjY0OGMiLz4="/>
```

Or when it is rejected:

```
<FlexRequestResponse
  Version="3.0.0"
  SenderDomain="aggregator.org"
  RecipientDomain="uftp.dso.nl"
  TimeStamp="2021-10-29T06:54:36.4437962Z"
  MessageID="7f0f4e68-f842-4b92-911e-b26f85525067"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  Result="Rejected"
  RejectionReason="Reference Period mismatch"
  FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"/>
```

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

```
<SignedMessage SenderDomain="aggregator.org" SenderRole="AGR"
               Body="+hM5FFliOQEpuMLwVpy6KKhb8OxOtrCu//VhLYnhiJAwZC8Eh743SPErLsPR9bpWTJdnuSfD
               SY3UamQuafGVCzxGbGV4UmVxdWVzdFJlc3BvbnNlDQogICAgICAgIFZlcnNpb249IjMuMC4wIg0KIC
               AgICAgICBTZW5kZXJEb21haW49ImFnZ3JlZ2F0b3Iub3JnIg0KICAgICAgICBSZWNpcGllbnREb21h
               aW49InVmdHAuZHNvLm5sIg0KICAgICAgICBUaW1lU3RhbXA9IjIwMjEtMTAtMjlUMDY6NTQ6MzYuND
               QzNzk2MloiDQogICAgICAgIE1lc3NhZ2VJRD0iN2YwZjRlNjgtZjg0Mi00YjkyLTkxMWUtYjI2Zjg1
               NTI1MDY3Ig0KICAgICAgICBDb252ZXJzYXRpb25JRD0iNDhjZGMzZDItNTZjMC00MzZjLThkNWEtNm
               Y2Y2MzZGM1MzhkIg0KICAgICAgICBSZXN1bHQ9IlJlamVjdGVkIg0KICAgICAgICBSZWplY3Rpb25S
               ZWFzb249IlJlZmVyZW5jZSBQZXJpb2QgbWlzbWF0Y2giDQogICAgICAgIEZsZXhSZXF1ZXN0TWVzc2
               FnZUlEPSJkM2FlNDgzNi01NWIxLTQwODQtYjU0ZS0zNDEwN2IyMjY0OGMiLz4="/>
```

FlexOffer

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

Example FlexOffer:

```
<FlexOffer
  Version="3.0.0"
  SenderDomain="agr.nl"
  RecipientDomain="dso.nl"
```

```
  TimeStamp="2021-10-29T06:54:36.8868538Z"
  MessageID="338ed243-5517-4400-962e-2b7b812c468c"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-30"
  CongestionPoint="ean.265987182507322951"
  ExpirationDateTime="2021-10-29T10:30:00Z"
  FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
  ContractID="A-AA-A-12345"
  BaselineReference=""
  Currency="EUR">
  <OfferOption OptionReference="ba40a5f8-849b-4fe6-958f-e628a1653558"
    Price="0.00">
    <ISP Power="50000000" Start="58"/>
    <ISP Power="50000000" Start="59"/>
    <ISP Power="50000000" Start="60"/>
    <ISP Power="50000000" Start="61"/>
  </OfferOption>
</FlexOffer>
```

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

```
<SignedMessage SenderDomain="agr.nl" SenderRole="AGR"
               Body="whfSzCmP9Oml2y/6lfz/7a6hRvz7Am5waAwmtKT52kbEG9OhqHTj4DisCRNnmGHrmDjAsq6AyNB/YG+gDnZ5BjxGbGV4
```

⚠️ 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.

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	⚠️ Same rules apply as ISPs in FlexRequest. 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:

```
<FlexOfferResponse
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="arg.nl"
  TimeStamp="2021-10-29T06:54:36.4437962Z"
  MessageID       = UUID
  ConversationID  = UUID
  ReferenceMessageID = UUID
  Result             = ("Accepted" | "Rejected")
  RejectionReason    = String (Only if Result = "Rejected")
/>
```

FlexOrder

Example FlexOrder:

```
<FlexOrder
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="agr.nl"
  TimeStamp="2021-10-29T06:55:36.518Z"
  MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-30"
  CongestionPoint="ean.265987182507322951"
  FlexOfferMessageID="338ed243-5517-4400-962e-2b7b812c468c"
  ContractID="A-AA-A-12345"
  Price="0.00"
  Currency="EUR"
  OrderReference="None">
  <ISP Power="50000000" Start="1" Duration="1"/>
  <ISP Power="50000000" Start="2" Duration="1"/>
  <ISP Power="50000000" Start="3" Duration="1"/>
  <ISP Power="50000000" Start="4" Duration="1"/>
</FlexOrder>
```

Attribute/Element	
Currency	Must be `EUR`.
Price	Must equal the Price in the FlexOffer.
OrderReference	May be filled by the calling grid company for settlement process. When making a call from the GOPACS UI, 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.

FlexOrderResponse

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

Example FlexOrderResponse:

```
<FlexOrderResponse
```

```
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="arg.nl"
  TimeStamp="2021-10-29T06:54:36.4437962Z"
  MessageID       = UUID
  ConversationID  = UUID
  ReferenceMessageID = UUID
  Result             = ("Accepted" | "Rejected")
  RejectionReason    = String (Only if Result = "Rejected")
/>
```

TestMessage & TestMessageResponse

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 and to test sending an receiving messages and message responses.

Currently, this is only supported for API, not on the GUI, it is not possible to trigger a test message on the GOPACS GUI.

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.

Other message types

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

GOPACS implementation specifics

⚠️ 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!

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

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

ℹ️ 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.

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