Flexibility trade (day-ahead) using own UFTP implementation

Last updated: 2026-04-01

With flexibility trade in congestion management is meant: *the trade in flexible power usage* between grid operators on the one hand and electricity consumers and producers on the other hand. For day-ahead, GOPACS currently only knows 'trade' via earlier made contracts between these parties, that determine when a grid operator can order the consumer/producer to reduce or increase its off-take/feed-in. Day-ahead flexibility trade in GOPACS consists of the activation of these capacity steering contracts (capaciteitssturende contracten, CSC) and alternative transport rights (alternatieve transportrechten, ATR).

For this, GOPACS use Shapeshifter UFTP. As a GOPACS user you can use GOPACS itself (via the website) for sending and receiving messages. But to have a better integration with your own applications, you can have your own implementation.

This manual is about how to create your own UFTP implementation. It will discuss the basics of UFTP communication and provide an extensive reference on the composition of the UFTP messages.

The role of UFTP participant determines which messages one sends and which ones are received. This manual focuses on the role of the (party representing the) consumer/producer and not on the grid operator.

Now, in order to actually to do flexibility trading or to test your own implementation, you will need to have a GOPACS account. You need to separately request accounts for production and acceptance. Please check GOPACS account aanmaken - GOPACS (production) and for more information.

Please note: in Q2 2026, GOPACS will introduce a pull API that allows you to easily retrieve the set limits yourself. This removes the need for a (complex) Shapeshifter UFTP implementation. If you have not yet started building, we recommend considering waiting for this and aligning your implementation accordingly.

Table of contents

Glossary

AGR	Aggregator, the term used in Shapeshifter UFTP to refer to any party representing an electricity consumer or producer.
DSO	Distributed System Operator, also known as operator of the (local) distribution grid. In Shapeshifter, the term is also applied to the Transmission System Operator.
Shapeshifter	Shapeshifter UFTP has become the name of UFTP after the original USEF foundation ceased to exist and version 3 was released.
UFTP	USEF Flexibility Trading Protocol
USEF	Universal Smart Energy Framework, for more information see the website of USEF.

For terms not appearing in the list above, please check the general GOPACS Glossary.

> A note about the terms off-take, feed-in, consumption and production. In the Shapeshifter specification the terms consumption and production are used to indicate the delivery directions on a connection. Within GOPACS we prefer to use the terms off-take and feed-in. Therefore, in this manual we the latter terms.

UFTP Basics

In this chapter we will provide an overview of what the USEF Flexibility Trading Protocol (UFTP) is about. More details are provided in later chapters.

We're using version 3.0.0 of UFTP for capacity steering contracts and version 3.1.0 for alternative transport rights. The messages used in UFTP communication are defined as XML Schema Definitions (XSD).

It be noted that UFTP from version 3 is referred to as Shapeshifter UFTP. For the sake of brevity, we will stick to 'UFTP' in this manual.

UFTP has been created with the idea of being able to set up an entire trading platform for flexibility trading. For now, this is not supported by GOPACS. Its use is limited to the activation of capacity steering contracts and alternative transport rights.

Signing keys

The Shapeshifter protocol dictates the use of two NaCl key pairs:

Purpose	NaCl function	Private key bits	Public key bits
Digital signatures	crypto_sign_keypair	512	256
Authenticated message encryption	crypto_box_keypair	256	256

The two public keys must be joined together in a single 64-byte array (signature key followed by encryption key), Base-64 encoded and prefixed by the constant cs1 and a period. For example, `cs1.V4lZrkYHq8FzneXxUML+QEMXMul3tBm+gPGoVDIZBA92VoWTu8/kRu2Zx72XOm1i/qwwoSgXjqSAqSa43myCaQ==`.

An easy way to generate the key is this small Java program.

For further information, please read the relevant Shapeshifter documentation

Roles

UFTP is about sending messages between grid operators and parties that represent electricity consumers or producers. UFTP distinguishes, therefore, two roles: **DSO** and **AGR**. DSO is distributed system operators or local grid operator. Within UFTP it also refers to Transmission System Operators. The parties, that represent the consumers or producers, are referred to as AGR (aggregators). For UFTP it does not matter whether they are really aggregators.

Asynchronous communication

UFTP communication is **asynchronous**: i.e. as long as the message is delivered to the right endpoint and can be read by the receiver, the receiver will respond with an OK (2XX) response. Asynchronously, the receiver will do the validation of the message and send a Response with either ACCEPTED or REJECTED

> ⚠ As a receiver you must 200 OK the request message (FlexRequest, FlexOffer, FlexOrder) before sending back a response message (FlexRequestResponse, FlexOfferResponse, FlexOrderResponse). Because many sending systems validate the referenced outgoing request message is fully processed before processing the incoming response message.

> ⚠ Consider a reasonable delay (e.g. at least 150 ms) between 200 OKing an incoming message and sending back a reply. This is because the upstream systems often have to commit database transactions before they are ready for the reply.

Message flow

The basic message flow is as follows:

- First of all, and out of scope of UFTP, a DSO determines when, where and who will be asked to reduce or increase its off-take or feed-in. It will result in a FlexRequest.
- Now, the DSO sends the **FlexRequest** to the AGR. (Remember that this protocol is asynchronous, so the AGR does no validation of the message yet. It will simply respond with an HTTP status 2XX)
- The AGR validates the content of the FlexRequest sends a **FlexRequestReponse** telling whether the FlexRequest was accepted or rejected.
- If the FlexRequest was accepted, the AGR subsequently sends a **FlexOffer** to the DSO.
- Now the DSO validates the FlexOffer and responds with a **FlexOfferResponse** telling whether the FlexOffer was accepted or rejected.
- Subsequently, if the FlexOffer was accepted, the DSO sends a **FlexOrder** AGR.
- And, finally, the AGR validates the FlexOrder and sends a **FlexOrderResponse**. Now the flexibility trade has been finalized.

All these messages are sent between DSO and AGR via the **GOPACS UFTP Message Broker**. That is, the message is delivered to the message broker with an identifier of both the sender and the recipient. The message broker delivers the message to the receiving party. For more information, see the next chapter.

Messages are encrypted with the private key of the sender and can be decrypted using the sender's public key. As a participant you will need to have a key pair in place.

For capacity steering contracts, the following applies: the FlexRequest sent by the DSO is leading. The FlexOffer must copy the capacity steering power value from the FlexRequest. And the final FlexOrder will do the same.

For alternative transport rights (like TDTR or VVTR) a simplification has been made in version 3.1.0. The simplification is that this version allows for so-called 'unsolicited' FlexOrders (i.e. a FlexOrder which is not preceded by a FlexRequest and FlexOffer). The AGR needs to have implemented version 3.1.0 of Shapeshifter UFTP in order to receive unsollicited FlexOrders. The DSO sends only a FlexOrder to the AGR and the AGR responds with a FlexOrderResponse.

Concerning the validation done by the AGR: FlexRequests or FlexOrders can only be rejected, if the request does not conform the terms of the contract between the DSO and AGR, or the message does not meet other criteria. (For further details, see the message specifications in FlexMessage catalog)

> With the coming of alternative transport rights and capacity deployment contracts, we're running into some limitations of Shapeshifter UFTP specification and the way it has been implemented in GOPACS. We're working on a next major version of Shapeshifter UFTP, in which these cases will be better accomodated.

UFTP Message Broker and address book

Message Broker

As mentioned above, all messages are sent via the Message Broker. In the table below you will find the endpoint for sending messages.

Now, for receiving messages, you need to expose an endpoint yourself to which the Message Broker can send messages. You may have to configure your firewall to allow request from the message broker to come in. The IPs from the message broker are also found in the table below.

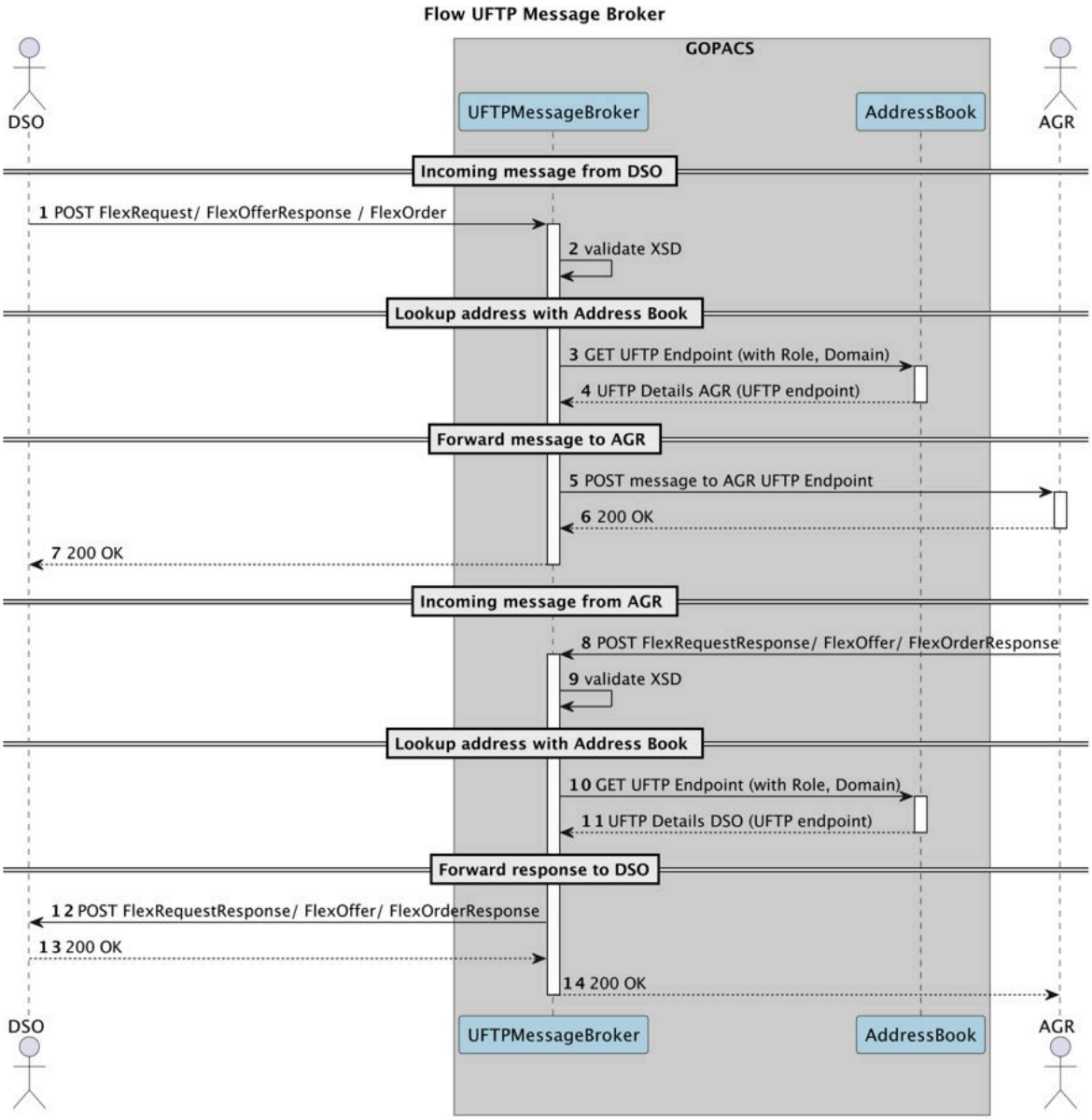
Optionally, you can secure your endpoint with OAuth2 and let GOPACS identify itself when delivering a message. In that case you need to provide the client credentials for the message broker and the URL of the authorization server in the Participant Details (see paragraph 'Participant details' further on).

The message broker requires authentication to be able to post messages. Once you have a GOPACS account, you can setup OAuth2 client credentials using the manual 'OAuth2 Client credentials for API Clients' found in the documentation section.

	Acceptance environment	Production environment
UFTP message endpoint	https://clc-message-broker.acc.gopacs-services.eu/shapeshifter/api/v3/message	https://clc-message-broker.gopacs-services.eu/shapeshifter/api/v3/message
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

Shapeshifter is originally set up for peer-to-peer communication. Now, the GOPACS UFTP Message Broker is required to be used as a broker for sending messages. The communication between sender and receiver is as follows. The sender posts the message, which contains who is to receive the message, to the Message Broker. In a single synchronous HTTP call, the broker decrypts the message using the sender's public key, validates the format of the message (XSD validation), looks up the endpoint of the receiver and sends the original, encrypted message to the receiver. After the Message Broker has received the HTTP response from the receiver, it sends a HTTP response to the sender. Hence, when the sender receives a HTTP OK from the Message Broker, the message has been validated against the XSD and has been delivered to the receiver. As UFTP communication is asynchronous, the receiver must confirm that he has accepted or rejected the message by sending a

response message. This response message is processed by the message broker as any other message. This flow is depicted in the below sequence diagram.



Now, during this process, the message delivery may fail for various reasons. In the table below the possible HTTP status codes and corresponding failures are described.

HTTP status	Possible cause

200	Message has correct signature, conformed to the message format defined in the XSD and has been successfully delivered to the receiver.
400	Wrong content type (must be text/xml) Error during XML deserialization The message did not pass the XSD validation Sender could not be found in the address book Receiver could not be found in the address book The receiver reject the message with an 4xx error.
401	Bearer token not provided or invalid. Public key of sender not found or incorrect. Message signature does not match with public key of sender.
403	Bearer token not authorized to perform this request.
409	Message with MessageID is already (being) forwarded A Response message is being sent, before the delivery of the original message has been finished.
500	Unexpected server error on GOPACS side.
502	The recipient was not configured correctly or returned a 5xx response
504	The recipient could not be reached

Participant API (address book)

In the previous chapter the flow of UFTP communication has been shown. For this communication some contact details are required. For this purpose, GOPACS has a Participant API. This Participant API has the following details:

1. the public key of a participant, which is needed to decrypt received message and to verify that they come from the sender

2. the 'UFTP domain' of a participant, which is needed for DSO's to tell the UFTP Message Broker who the recipient is.

For more information on this API, please read the OpenAPI documentation found via the links mentioned in the table below.

	Acceptance environment (for testing)	Production environment
Participant API endpoint	https://clc-message-broker.acc.gopacs-services.eu/v2/participants/	https://clc-message-broker.gopacs-services.eu/v2/participants/
OpenAPI documentation (Swagger)	Swagger UI of Swagger UI	Swagger UI of Swagger UI

NB! The Participant API returns a Participant-object containing a third property, which is not mentioned above. This is the 'endpoint' of the participant. This is a remnant of when UFTP messages were sent peer-to-peer. As messages are now sent via the message broker, this property is no longer relevant. (It now always contains the endpoint of the UFTP Message Broker.)

Participant details

Every UFTP participant must have its contact details registered in GOPACS. Some of those details are available to other participants via the Participant API. Other are used by the message broker to deliver messages.

If you have your own implementation, you will need to have:

- a '*UFTP domain*', which is an identifier used within messages. It does not have to be your actual internet domain, but it could be. It is not used for routing.
- the *public key* with which receivers of your messages can decrypt them
- the *endpoint* to which the message broker must deliver messages

Now, *if you want the message broker to authenticate when delivering messages* to your endpoint, you must provide the client credentials, token endpoint and, if applicable, the 'scope'. **NB**! This is not to be confused with the client credentials your application needs to deliver messages to the message broker.

In the manual 'Setting UFTP Participant details' in the documentation section you can find a step by step guide to fill in these details.

Open source library

There is a Shapeshifter UFTP library available as open source, which you can use to aid in the development of your own UFTP implements. There is Java library and a Python library available.

The Shapeshifter Java Library dependencies are available in the Maven repository: https://mvnrepository.com/artifact/org.lfenergy.shapeshifter. The source code is available at https://github.com/shapeshifter/shapeshifter-library-java.

For the Python Library, see https://github.com/shapeshifter/shapeshifter-library-python.

Central concepts in FlexMessages

There are few central concepts in FlexMessages, which are explained in this chapter. And, as mentioned in the chapter 'UFTP Basics', version 3.1.0 deviates slightly, which will also be addressed in this chapter.

Sender and recipient domain and role

Each FlexMessage has the properties **SenderDomain** and **RecipientDomain**. These refer to the 'UFTP domain' property that is part of the UFTP Participant details (see chapter UFTP Message Broker and address book). This property is used as an identifier within UFTP communication. It be noted that for capacity steering contracts that allows cross-operator activation, the sender of the FlexRequest or FlexOrder does not have to be the contract partner.

When you receive a message, you receive a SignedMessage which you will have to decrypt. You need to read the **SenderDomain and SenderRole** properties, to obtain the sender's public key from the Participant API and decrypt the SignedMessage.

The SenderRole is either DSO or AGR. If you are an AGR, you will only receive messages from a sender with SenderRole=DSO and you will only be sending messages with SenderRole=AGR.

ISP, TimeZone and Period

Central in flexbility trade is the concept of **Imbalance Settlement Period (ISP)**, which is a period in which there is congestion. For these periods, the AGR is requested to increase or decrease its off-take or feed-in.

ISPs are bound to a **TimeZone**, but TimeZone is defined as a property of the message the ISPs belong to. Within GOPACS, the time zone is always "Europe/Amsterdam".

The **ISP-Duration** defines the standard length of an ISP. Within GOPACS, ISP-Duration is always "*PT15M*", meaning that ISPs have a **fixed length of 15 minutes**.

This makes that most days have 96 ISPs, with an exception of the days Daylight Savings Time starts or ends. Each period of 15 minutes has a **1-based 'index'**.

- ISP 1 is 00:00:00 (inclusive) until 00:15:00 (exclusive)
- ISP 2 is 00:15:00 (inclusive) until 00:30:00 (exclusive)
- Last ISP of the day is for most days 96 and is 23:45:00 (inclusive) until 00:00:00 the next day (exclusive)

Now, with regard to **Daylight Saving Time**. As the time zone is always Europe/Amsterdam:

- 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
  et cetera
  ISP 8: 01:45-**03:00**
  ISP 9: **03:00**-03:15
  et cetera
  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
  et cetera

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
et cetera
ISP 100: 23:45-00:00

ISPs appear in the message types FlexRequest, FlexOffer and FlexOrder. They have slightly differing properties in these message types. More information about those is found in the description of those message types further on in this chapter. But they have the following **common properties**:

- **Duration**, is the length of this ISP expressed in number of periods of 15 minutes. If duration is 1, this ISP has a length of 15 minutes. If duration is 8, it has a duration of 2 hours.
- **Start**, refers to the 1-based index of this ISP in the day. That is, if start is 1, the ISP starts at 00:00. If start is 6, the ISP starts at 01:15. And so on.

Some examples, all on days on which there is no change of Daylight Savings Time. If start = 25, duration = 1, the the ISP refers to 06:00 - 06:15. If start = 8, duration = 4, then the ISP refers to 01:45 - 02:45.

Now, whereas the ISP contains information about the time, at message levelt there is a **Period** property which defines the date. This means that the ISPs in a single message are restricted to a single date. If a DSO wants to send a FlexRequest for a period spanning, for example, from 2025-11-15 at 23:00:00 until 2025-11-16 at 01:00, he will have to split this request into two FlexRequests for 15 and 16 November.

ISP Power (MinPower, MaxPower)

In a FlexRequest the ISPs have a **MinPower** and a **MaxPower** attribute. In a FlexOffer and FlexOrder, they have a single **Power** attribute. These values refer to the capacity to which the AGR is to raise or lower its off-take/feed-in.

- The value is in Watt. Only multiples of 1000 are allowed.
- A positive value refers to off-take
- A negative value refers to feed-in

⚠ If you have read the Shapeshifter specification, you will notice an important difference with regard to the power attributes between the Shapeshifter specification and the GOPACS implementation. Whereas in the Shapshifter specification they are a deviation from a baseline (*drop-by*), in GOPACS the power attributes refer to an absolute capacity (*drop-to*).

MinPower and MaxPower values in a FlexRequest for the 4 capacity steering types.

Contract Steering Capacity distinguishes two types of capacity steering. The capacity can be *limited*, in which the AGR is ordered not to consume or produce more electricity than the given capacity. And, second, the capacity can be *deployed,* in which the AGR is orderder to consume or produce at least as much as the given capacity. Alternative transport rights only concern limitation, see Unsolicited FlexOrder version 3.1.0 for TDTR and VVTR.

With two types of steering for the two delivery directions off-take and feed-in, there are four types of capacity steering: off-take limitation, off-take deployment, feed-in limitation and feed-in deployment.

Within the context of capacity steering for congestion management, a limitation is understood as that the capacity use is supposed to be limited at least to the given capacity and it may be lower. A deployment, vice versa, is understood as that the capacity is supposed to be increased to at least the given capacity and it may be higher. (It be noted that *grid balance* may be a responsibility of the AGR as weel, but it is ignored in the context of congestion management.)

Now, the values of MinPower and MaxPower in a FlexRequest indicate a range within which the capacity use is ordered to be. This range consists of the **steering value** and a value that indicates the *opposit border*. In the case of:

- **feed-in limitation:**
  - `MinPower` is the steering value, that is the maximum allowed feed-in (feed-in, hence a negative number). E.g. when limiting the feed in to 3 MW, `MinPower` = -3000000.
  - `MaxPower` = 0
- **off-take limitation**:
  - `MaxPower` is the steering value, the maximum allowed off-take in Watt.
  - `MinPower` = 0
- **feed-in deployment**:

    - `MaxPower` is the steering value, the minimum required feed-in (a negative number).E.g. if the producer is ordered to feed-in at least 20MW, maxPower = -20000000.
    - `MinPower` is the maximum feed-in a producer can give according to the grid operator, which, in principle, is the contracted transport capacity (as a negative number). So, if the producer has a contracted transport capacity for the feed-in direction of 100MW, minPower = -100000000.
- **off-take deployment**:
    - `MinPower` is the steering value, the minimum required off-take.
    - `MaxPower` is the maximum off-take a consumer can take according to the grid operator, which, in principle, is the contracted transport capacity.

Notes on the above:

- What matters, is that the receiver of a message knows what the **steering value** is and whether there is a limitation or a deployment. The steering value is what needs to be taken over from the FlexRequest in the FlexOffer.
    - A simple rule for deriving the steering value: if one of MinPower and MaxPower is 0, it is the value that is not 0. If none of them is 0, it is the value that is closest to 0.
- The ‘opposite border’ value is supposed to be understood only as to tell whether it is a limitation or a deployment and is not to be taken strictly.
    - For instance, an off-take limitation FlexRequest, strictly speaking, is saying that the power use should be between 0 and a (positive) steering value indicating the maximum off-take. However, it is perfectly allowed to feed in electricity when you have received an off-take limitation. That means, it is allowed to go lower than the minPower. The same applies for a feed-in limitation, in which you are perfectly allowed to take off electricity.
    - For a deployment, the opposit boundary will typically be the contracted transport capacity, but it does not have to be.
- Both MinPower and MaxPower could be 0 to completely limit both off-take and feed-in.
- The values for MinPower and MaxPower must always conform to the agreements in the capacity steering contract or alternative transport right. Also, these values should not be higher than the contracted transport capacities.

ConversationID and MessageID

FlexMessages have an ID of type UUID. These IDs are used for every follow-up message to refer to the message, which they are following up (see the reference of each message type).

Next to that, messages are part of a **conversation**. A conversation is the collection of messages referring to the same conversationId (type UUID). The conversationId is determined by the creator of a FlexRequest and each following message must use the same conversationId. A conversation has no own properties.

Timestamp

Each message must have timestap for indicating when this message was created. It must be formatted according to ISO8601 including the offset. The offset can be either in “+HH:mm:ss” format or “Z”. The time zone or offset in this timestamp does not have to be the same as the TimeZone property of the message.

FlexMessage catalog

In this chapter the various types of messages and the relations between them are explained. In the Shapeshifter API library you will find the XSD schemas defining all the message types. They can also be found in the Shapeshifter Specification project on GitHub as source code.

SignedMessage

All messages are sent encrypted and wrapped in a SignedMessage. If you are not using one of the provided open source libraries for your implementation, please read Shapeshifter's manual on 'Cryptographic scheme' for more details on encryption.

Here is an example of a SignedMessage HTTP request (some headers omitted for clarity):

```
POST /shapeshifter/api/v3/message HTTP/1.1
Accept: application/json (or */* or omit)
Authorization: Bearer ...
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SignedMessage
  SenderDomain="dso.nl"
  SenderRole="DSO"
Body="SwGOfSa4bZ9ghmuQmm7lvTkgDvF2F/dy1A3qqe7qkciiH/qyIuXdAAxfV8+jqW8Gc91pcqMoYr8
```

Attribute	GOPACS expectation
SenderDomain	The UFTP domain property of the sender. See paragraph Sender and recipient domain and role.
SenderRole	Must be `DSO` for a grid operator and `AGR` for a (party representing a) consumer or producer. See paragraph Sender and recipient domain and role. GOPACS is not considered a 'party' in the UFTP exchange and therefore does not have a role.
Body	The Body contains the actual message encoded and encrypted. The receiver needs the SenderDomain SenderRole properties to obtain the sender's public key from the Participant API and decrypt the message. (And thereby verify the origin of the message.) If you are not using one of the provided open source libraries for your implementation, please read Shapeshifter's manual on 'Cryptographic scheme' for more details on encryption. (Or download the open source code and learn how it is done there.)

FlexRequest

For capacity steering contracts each contract activation starts with a FlexRequest (see Message flow) from the DSO and this FlexRequest is leading in the sense that the subsequent FlexOffer must conform the values in the FlexRequest.

Here is an example of a FlexRequest for the activation of a Capacity Steering Contract, where an off-take limitation to 50 MW is requested for 2021-09-29 from 11:45 to 12:45:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexRequest
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="agr.nl"
  TimeStamp="2021-09-29T06:54:26Z"
  MessageID="d3ae4836-55b1-4084-b54e-34107b22648c"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ServiceType="CBC"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-01"
  ContractID="A-AA-A-12345"
  CongestionPoint="ean.265987182507322951"
  Revision="1"
```

```
  ExpirationDateTime="2021-09-30T12:00:00.0000Z">
  <ISP Start="48" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500
  <ISP Start="49" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500
  <ISP Start="50" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500
  <ISP Start="51" Duration="1" Disposition="Requested" MinPower="0" MaxPower="500
</FlexRequest>
```

<table>
<tr><th>Attribute/Element</th><th>GOPACS additional restrictions</th></tr>
<tr><td>Version</td><td>Must be <code>3.0.0</code> (currently).</td></tr>
<tr><td>SenderDomain</td><td>See paragraph Sender and recipient domain and role.</td></tr>
<tr><td>RecipientDomain</td><td>See paragraph Sender and recipient domain and role.</td></tr>
<tr><td>TimeStamp</td><td>See paragraph Timestamp</td></tr>
<tr><td>MessageID</td><td>Required, type UUID. See ConversationID and MessageID</td></tr>
<tr><td>ConversationID</td><td>Required, type UUID. See ConversationID and MessageID</td></tr>
<tr><td>Revision</td><td>Required. Must be <code>1</code>. Revisions are currently not supported yet.</td></tr>
<tr><td>ServiceType</td><td>Optional. The ServiceType specifies which type of flexibility is requested. Currently the following values are supported:
<table>
<tr><th colspan="2">ServiceType</th></tr>
<tr><td>CBC</td><td>Capacity Steering Contract (abbreviation of the old name CapaciteitsBeperkend Contract). Will be replaced soon with ‘CSC’.</td></tr>
<tr><td>TDTR</td><td>Time bound transport right (a type of alternative transport right)</td></tr>
<tr><td>VVTR</td><td>Non-firm transport right (a type of alternative transport right)</td></tr>
</table></td></tr>
<tr><td>ISP-Duration</td><td>Required. Value must always be <code>PT15M</code>. See ISP, TimeZone and Period.</td></tr>
</table>

TimeZone	Required. Must be `Europe/Amsterdam`. See ISP, TimeZone and Period.
Period	Required. The date of the congestion problem. Format: `YYYY-MM-DD`. See ISP, TimeZone and Period.
ExpirationDateTime	Required. The dateTime until which the FlexRequest is valid. The entire conversation must be finalized before the ExpirationDateTime has been reached. The ExpirationDateTime must be no later than 12:00:00 the day before Period.
ContractID	Required. The ID of the capacity steering contract that is being activated.
CongestionPoint	Required. In the context of GOPACS this refers to a grid connection and it must be the grid connection with which the contract, specified by ContractID, deals. The grid connection is referred to by its EAN in the format `ean.[0-9]{18}`, e.g. `ean.265987182507322951`.
ISP	At least one is required. See See ISP, TimeZone and Period.
ISP.Start ISP.Duration	Required. See ISP, TimeZone and Period.
ISP.Disposition	Required. In the GOPACS context only `Requested` is a valid value. Can be ignored by AGR.
ISP.MinPower ISP.MaxPower	Required. See ISP Power (MinPower, MaxPower).

FlexRequestResponse

The FlexRequestReponse must be sent by the AGR after having received and validated the FlexRequest and tells whether the FlexRequest is accepted or rejected.

In the context of capacity steering contracts, the AGR is not allowed to decline a FlexRequest as long as it conforms to the terms in the contract. Only if the message is malformed or the FlexRequest does not meet the terms in the contract, is the AGR allowed to reject it.

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

```
<FlexRequestResponse
  Version="3.0.0"
  SenderDomain="agr.nl"
  RecipientDomain="dso.nl"
  TimeStamp="2021-09-29T06:54:36Z"
  MessageID="7f0f4e68-f842-4b92-911e-b26f85525067"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  Result="Rejected"
  RejectionReason="CongestionPoint is unknown"
  FlexRequestMessageID="d3ae4836-55b1-4084-b54e-34107b22648c"/>
```

Attribute/Element	GOPACS additional restrictions
Version	Must be `3.0.0` (currently).
SenderDomain	See paragraph Sender and recipient domain and role.
RecipientDomain	See paragraph Sender and recipient domain and role.
TimeStamp	See paragraph Timestamp. Make sure the created timestamp is later than the timestamp of previous, received messages.
MessageID	Required, type UUID. See ConversationID and MessageID
ConversationID	Required, type UUID. Must be the same as the ConversationID on the FlexRequest. See ConversationID and MessageID.
Result	Required. Allowed values are `Accepted` and `Rejected`. Only if the message is malformed or the FlexRequest does not meet the terms in the contract, is the AGR allowed to reject it.
RejectionReason	Required only if `Result=”Rejected”`. The reason why the FlexRequest was rejected.
FlexRequestMessageID	Required, type UUID. The MessageID of the FlexRequest to which this message responds.

FlexOffer

After having sent the FlexRequestReponse, a FlexOffer is sent as well by the AGR in reponse to the FlexRequest. The AGR sends only one FlexOffer in response to the FlexRequest, unless the FlexOffer was rejected by the DSO in the FlexOfferResponse.

Example of a FlexOffer message for a capacity steering contract:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexOffer
  Version="3.0.0"
  SenderDomain="agr.nl"
  RecipientDomain="dso.nl"
  TimeStamp="2021-09-29T06:54:46Z"
  MessageID="338ed243-5517-4400-962e-2b7b812c468c"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-01"
  CongestionPoint="ean.265987182507322951"
  ExpirationDateTime="2021-09-30T12:00:00.0000Z"
  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 Start="48" Duration="1" Power="50000000"/>
    <ISP Start="49" Duration="1" Power="50000000"/>
    <ISP Start="50" Duration="1" Power="50000000"/>
    <ISP Start="51" Duration="1" Power="50000000"/>
  </OfferOption>
</FlexOffer>
```

Attribute/Element	GOPACS additional restrictions
Version	Must be `3.0.0` (currently).
SenderDomain	See paragraph Sender and recipient domain and role.
RecipientDomain	See paragraph Sender and recipient domain and role.
TimeStamp	See paragraph Timestamp. Make sure the created timestamp is later than the timestamp of previous, received messages.
MessageID	Required, type UUID. See ConversationID and MessageID.
ConversationID	Required, type UUID. Must be the same as the ConversationID on the FlexRequest. See ConversationID and MessageID.
ISP-Duration	Required. Value must always be `PT15M`. See ISP, TimeZone and Period
TimeZone	Required. Must be `Europe/Amsterdam` See ISP, TimeZone and Period

Period	Required. The date of the congestion problem. This value is copied from the FlexRequest. See ISP, TimeZone and Period.
ExpirationDateTime	Required. The dateTime until which the FlexOffer is valid. This value must be copied from the FlexRequest.
FlexRequestMessageID	Required, type UUID. The MessageID of the FlexRequest to which this message responds.
ContractID	Required. The ID of the capacity steering contract that is being activated. This value is copied from the FlexRequest.
CongestionPoint	Required. In the context of GOPACS this refers to a grid connection and it must be the grid connection with which the contract, specified by ContractID, deals. The grid connection is referred to by its EAN in the format `ean.[0-9]{18}`, e.g. `ean.265987182507322951`. This value is copied from the FlexRequest.
BaselineReference	Ignored, can be left empty.
Currency	Required, but will be ignored. Use a valid value, like `EUR`.
OfferOption	Exactly 1 OfferOption element is expected.
OfferOption.OptionReference	Required, type UUID. A unique reference for this OptionReference. Can be referenced to in the FlexOrder, but in the context of capacity steering contracts it is not used.
OfferOption.Price	Required, but ignored. Must comply with `w ISO 4217`. `0.00`, `0.0` and `0` are all allowed.
OfferOption.MinActivationFactor	Optional. Ignored.
OfferOption.ISP	Take over the ISPs from the FlexRequest with the Start and Duration attributes.
OfferOption.ISP.Power	Required. The Power attribute must be the *steering value* from the corresponding ISP in the FlexRequest. Please read ISP Power (MinPower, MaxPower) for a description of the steering value.

FlexOfferResponse

In response to the FlexOffer, the DSO sends a FlexOfferResponse after he has received the FlexOffer from the AGR. Only if the FlexOffer is malformed or deviates from the FlexRequest, should the FlexOffer be rejected.

Here is an Example FlexOfferResponse:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexOfferResponse
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="arg.nl"
  TimeStamp="2021-10-29T06:54:36.4437962Z"
  MessageID="e240d10e-cd53-477f-b322-63dd98c03e94"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  FlexOfferMessageID="338ed243-5517-4400-962e-2b7b812c468c"
  Result="Accepted"
/>
```

For the explanation of the attributes of a FlexOfferResponse, please consult FlexRequestResponse. The only difference is that this Response type has a FlexOfferMessageID referring to the FlexOffer, rather than a FlexRequestMessageID.

FlexOrder

In the message flow for capacity steering contracts, the DSO sends in response to a FlexOffer a FlexOfferReponse and then a FlexOrder. In the message flow for alternative transport rights, the DSO will send a FlexOrder right away without there being a FlexRequest and FlexOffer.

Here is an example of a FlexOrder message for a CSC contract:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexOrder
  Version="3.0.0"
  SenderDomain="dso.nl"
  RecipientDomain="agr.nl"
  TimeStamp="2021-09-29T06:54:56Z"
  MessageID="dc0f19c4-3835-4753-8f0c-0319d6642fbb"
  ConversationID="48cdc3d2-56c0-436c-8d5a-6f6cc3dc538d"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2021-10-01"
  CongestionPoint="ean.265987182507322951"
  FlexOfferMessageID="338ed243-5517-4400-962e-2b7b812c468c"
  ContractID="A-AA-A-12345"
  Price="0.00"
  Currency="EUR"
  OrderReference="None">
  <ISP Start="48" Duration="1" Power="50000000/>
  <ISP Start="49" Duration="1" Power="50000000" />
  <ISP Start="50" Duration="1" Power="50000000"/>
  <ISP Start="51" Duration="1" Power="50000000"/>
</FlexOrder>
```

Attribute/Element	GOPACS additional restrictions
Version	Is `3.0.0` for CSC.
SenderDomain	See paragraph Sender and recipient domain and role.
RecipientDomain	See paragraph Sender and recipient domain and role.
TimeStamp	See paragraph Timestamp. Make sure the created timestamp is later than the timestamp of previous, received messages.
MessageID	Required, type UUID. See ConversationID and MessageID.
ConversationID	Required, type UUID. Must be the same as the ConversationID on the FlexRequest and FlexOffer. See ConversationID and MessageID.
ISP-Duration	Required. Value must always be `PT15M`. See ISP, TimeZone and Period
TimeZone	Required. Must be `Europe/Amsterdam` See ISP, TimeZone and Period
Period	Required. The date of the congestion problem. This value is the same for FlexRequest and FlexOffer. See ISP, TimeZone and Period.
FlexOfferMessageID	Required, type UUID. The MessageID of the FlexOffer to which this message responds.
ContractID	Required. The ID of the capacity steering contract that is being activated. This value is the same for FlexRequest and FlexOffer.
CongestionPoint	Required. This value is the same as on FlexRequest and FlexOffer.
BaselineReference	Ignored.
Currency	Required, but can be ignored.
OrderReference	Optional, type UUID. May be filled in by the DSO for the settlement process.

OptionReference	Optional, type UUID. If used, it must refer to an OfferOption of the FlexOffer to which this FlexOrder responds. In the context of capacity steering contracts it is ignored.
Price	Required, but can be ignored. Must comply with ISO 4217. `0.00`, `0.0` and `0` are all allowed.
ActivationFactor	Can be ignored.
ISP	The ISPs should match the ISPs in the FlexRequest and be identical to ISPs in the FlexOffer. See ISP, TimeZone and Period and ISP Power (MinPower, MaxPower) for more information.

Unsolicited FlexOrder version 3.1.0 for TDTR and VVTR

As mentioned above, the message flow for alternative transport rights deviates from capacity steering contracts. The DSO sends an unsolicited FlexOrder only.

The AGR must have version 3.1.0 of Shapeshifter UFTP implemented. The differences with version 3.0.0 are few and are mentioned below.

Here is an example of a FlexOrder message for a TDTR contract, where the transport capacity is limited to 50 MW:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexOrder
  Version="3.1.0"
  SenderDomain="dso.nl"
  RecipientDomain="agr.nl"
  Unsolicited="true"
  TimeStamp="2025-11-25T09:10:36.518Z"
  MessageID="8ee32857-6c7c-42b0-a2cd-7cfc37200b96"
  ConversationID="11602d97-87e2-4b7d-875f-35592c5417a6"
  ServiceType="TDTR"
  ISP-Duration="PT15M"
  TimeZone="Europe/Amsterdam"
  Period="2025-11-26"
  CongestionPoint="ean.265987182507322951"
  ContractID="0000001"
  Price="0.00"
  Currency="EUR"
  OrderReference="None">
  <ISP Start="58" Duration="1" Power="50000000"/>
  <ISP Start="59" Duration="1" Power="50000000"/>
  <ISP Start="60" Duration="1" Power="50000000"/>
  <ISP Start="61" Duration="1" Power="50000000"/>
</FlexOrder>
```

In the table below only those attributes are mentioned that deviate from version 3.0.0.

<table>
<tr><th>Attribute/Element</th></tr>
<tr><td>Version</td><td>Must be <code>3.1.0</code>.</td></tr>
<tr><td>ServiceType</td><td>Specify the type of ATR contract from one of the values in the table below:<table><tr><th colspan="2">ServiceType</th></tr><tr><td>TDTR</td><td>Time-bound transport right (tijdsduurgebonden transportright)</td></tr><tr><td>VVTR</td><td>Non-firm transport right (Volledig Variabel Transportrecht)</td></tr></table></td></tr>
<tr><td>ContractID</td><td>The ID of the alternative transport right that is being activated.</td></tr>
<tr><td>ISP</td><td>ISPs that are to be limited under the contract. See See ISP, TimeZone and Period and ISP Power (MinPower, MaxPower) for more information.</td></tr>
<tr><td>ISP.Power</td><td>The capacity to which the AGR is limited. See ISP Power (MinPower, MaxPower) for more information.</td></tr>
</table>

FlexOrderResponse

When the AGR receives a FlexOrder, he will respond with a FlexOrderResponse. Only when the FlexOrder is malformed or does not conform the terms of the contract or preceding FlexRequest, the AGR is allowed to reject the FlexOrder. After having sent the FlexOrderResponse, the FlexOrder is final and the Flex conversation is finalized.

Here is an example FlexOrderResponse:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlexOrderResponse
  Version="3.0.0"
  SenderDomain="agr.nl"
  RecipientDomain="dso.nl"
  TimeStamp="2025-11-25T09:10:38Z"
```

```
  MessageID="e241cfcc-71db-4108-bc1a-7b3f34d06884"
  ConversationID="11602d97-87e2-4b7d-875f-35592c5417a6"
  FlexOrderMessageID="8ee32857-6c7c-42b0-a2cd-7cfc37200b96"
  Result="Accepted"
/>
```

For the explanation of the attributes of a FlexOrderResponse, please consult FlexRequestResponse. The only difference is that this Response type has a FlexOrderMessageID referring to the FlexOrder, rather than a FlexRequestMessageID.

Testing a UFTP implementation

Acceptance environment

On the acceptance environment, we have created a UFTP Testing functionality to test as an AGR whether your UFTP implementation properly receives and sends flex messages. Please read the manual **Testing a UFTP implementation using the UFTP Testing functionality** in Documents and manuals.

Production environment

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