**API EDI.Kontur documentation**

EDI.Kontur is a service for electronic exchange of standardized messages and documents between suppliers and customers.

Its HTTP-API interface is a baseline for integration with EDI.Kontur. This baseline is considered the most common which, in particular, provides platform independence for integration achievements.

It means that either clients written in C# language for .NET platform and starting on machines running Microsift Windows OS or clients written in JAVA or C++ starting on the machines running Linux OS are able to work with HTTP-API.

**Contents:**

* Data model
* API processing
* Supporting tasks
* Data structures
* Enumerations

**Revision history:**

* API revision history

**Data Model**

**Boxes** (box) are routing items in the EDI.Kontur system. When a user intends to send an EDI-document to their counterparty, they create a document in their box (we shall call it **a** **sender’s box**) and the system delivers it by forming correspondent document in the counterparty box (**a recipient’s box**). The box can be customized the following way: after a document has appeared in the box, its physical transmission occurs via some protocol (e.g. FTP, AS2).

Each box in the EDI.Kontur system refers to an organization (a legal entity or an entrepreneur). The boxes are attached to organizations so that each box belongs to one and only one organization.

EDI.Kontur users can be authenticated using username and password. Access control of authenticated users is provided on the box-level. It is possible that several users have access to one box and vice versa that one user has access to several boxes. User access is the ability to see the documents in the box and send them from the box to the counterparty boxes.

In the EDI.Kontur system exchange of documents is provided in the form of messages. A box is a logical container holding all messages ever being sent from and to the box. Event mechanism is used to see all the messages inside the box and to monitor their status. The box can be descried as a continuous event stream where each event refers to some message. The list of the main kinds of the events is provided below:

* a new inbound message is delivered;
* a new outbound message is sent;
* an outbound message is delivered to the counterparty box;
* an outbound message is not delivered to the counterparty box (due to an error);
* an outbound message is read by the counterparty;
* an outbound message has passed the check on the recipient’s side;
* an outbound message has not passed the check on the recipient’s side;
* tax invoice, TORG-12 delivery note drafts are formed in accordance with an outbound message Invoic;
* tax invoice, TORG-12 delivery note drafts formed in accordance with an outbound message Invoic are signed and sent via Diadoc;
* tax invoice, TORG-12 delivery note drafts formed in accordance with an outbound message Invoic are deleted in Diadoc;
* a tax invoice, a TORG-12 delivery note formed in accordance with an outbound message Invoic are signed by the sender and delivered via Diadoc;
* mistakes in the tax invoice, TORG-12 delivery note formed in accordance with an outbound Invoic have been found on the sender’s side;
* a tax invoice, a TORG-12 delivery note have been canceled.

All told, to integrate the system it is required to provide information about new events in the box and respond to them (e.g. to form reply messages or to show status of sent messages).

**API processing**

Authentication

Most API commands require authentication. As a required parameter, commands need so-called security token – a byte array unambiguously identifying a user. Besides, it is required to pass so-called “developer key”, a unique string identifier of an integrator, in all interface methods of integration. The integrator must not pass its “developer key” to a third party.

Authorization is a standard HTTP Header used to pass authorizing information with each request. Authentication scheme used by the EDI.Kontur system is called KonturEdiAuth. Its parameters are the following:

* **konturediauth\_api\_client\_id** serves to pass a developer key;
* **konturediauth\_token** serves to pass a security token;
* **konturediauth\_login** serves to pass a username (used only at the moment of receiving the security token);
* **konturediauth\_password** serves to pass a password (used only at the moment of receiving the security token).

Parameter values and their names are separated by the carachters ‘=’. Parameters are separated by ‘,’. For example:

Authorization: KonturEdiAuth konturediauth\_api\_client\_id=testClient-8ee1638deae84c86b8e2069955c2825a, konturediauth\_token=3IU0iPhuhHPZ6lrlumGz4pICEedhQ1XmlMN1Pk8z0DJ51MXkcTi6Q3CODCC4xTMsjPFfhK6XM4kCJ4JJ42hlD499/Ui5WSq6lrPwcdp4IIKswVUwyE0ZiwhlpeOwRjNrvUX1yPrxr0dY8a0w8ePsc1DG8HAlZce8a0hZiWylMqu23d/vfzRFuA==

So that the overall action sequence required while accessing to the functions of the integrator interface is the following:

1. To receive a security token after having requested the server and having passed the developer key as well as a username and password by using the command Authenticate.
2. In all other methods, it is required to pass the security token as well as the developer key.

**Important**

It is not necessary to authenticate before each request. A security token is valid up to 12 hours so cache it. If the token gets invalid, the server sends 401 HTTP code (Unauthorized). In this case, repeat Authenticate request and use a new token.

Presettings

Before using EDI-messages, check and save the data that will be needed later:

**GetAccessiblePartiesInfo:**

This method returns the information about organizations available for a user. Organization identifier **partyld** is used as an input value in the following method and in the methods used for receiving information about an organization: GetPartyInfo, GetUsersInfo, GetOrganizationCatalogueInfo.

**GetMainApiBox:**

This method returns **boxld** – an organization box identifier. This identifier you will use as a required parameter while sending messages (SendMessage method) and reading events in the box (GetEvents method).

**GetBoxDocumentsSettings:**

This method returns information about the messages available for receiving and sending on each of the organization counterparties.

Received data change in exceptional circumstances (e.g. in case of GLN or message sequence with a partner change), so there is no need to request them regularly.

Single API access scheme

1. **Authorization:** get a security token, save it and use it in following requests in the Authorization header.
2. **Message sending:** send messages using SendMessage method.
3. **Receiving new events:** new events processing in the box is done in several steps:
   1. **Receive new events using GetEvents method, in exclusiveEventld specify:**

* The identifier of the last processed event from **BoxEventBatchlastEventld** if you have requested events before.
* The **EventPointer** identifier of the last processed event if you have to start processing of the event queue since the exact event in the list.
* Null value if you have to get the list of the events since the very beginning of work.

You will receive the list of new events and the identifier of the last processed event **lastEventld**.

* 1. Update the current condition of outbound messages according to the data in the corresponding events.
  2. Receive inbound messages using **GetInboxMessage** method for all the events of *NewInboxMessage* kind.
  3. Send **GetEvents** request again, specifying the identifier of the last processed event **lastEventId** in **exclusiveEventId**. Repeat until you can read the list to the end, i.e. the list of the receivied events is empty.
  4. Save **lastEventld** and use it to receive events in the following session.

**Supporting tasks**

In many commands JSON serialization objects perform communication between a client and the server. We shall call them with a term *structure*. I.e. if a document says that the server has returned the structure like

**class** **someClass** {

someStringProperty: string,

someIntProperty: int,

someSubClassProperty: {

someSubClassStringProperty: string,

}

}

it means that server response will be the following:

{**"SomeStringProperty"**:"string1",**"SomeIntProperty"**:12,**"SomeSubClassProperty"**:{**"SomeSubClassStringProperty"**:"string2"}}

**Supporting tasks**

* [Authenticate](http://docs.edi.kontur.ru/en/latest/methods/Authenticate.html)
* [GetAccessiblePartiesInfo](http://docs.edi.kontur.ru/en/latest/methods/GetAccessiblePartiesInfo.html)
* [GetBoxDocumentsSettings](http://docs.edi.kontur.ru/en/latest/methods/GetBoxDocumentsSettings.html)
* [GetBoxesInfo](http://docs.edi.kontur.ru/en/latest/methods/GetBoxesInfo.html)
* [GetEvents](http://docs.edi.kontur.ru/en/latest/methods/GetEvents.html)
* [GetEventsFrom](http://docs.edi.kontur.ru/en/latest/methods/GetEventsFrom.html)
* [GetInboxMessage](http://docs.edi.kontur.ru/en/latest/methods/GetInboxMessage.html)
* [GetMainApiBox](http://docs.edi.kontur.ru/en/latest/methods/GetMainApiBox.html)
* [GetOrganizationCatalogueInfo](http://docs.edi.kontur.ru/en/latest/methods/GetOrganizationCatalogueInfo.html)
* [GetOutboxMessage](http://docs.edi.kontur.ru/en/latest/methods/GetOutboxMessage.html)
* [GetPartyInfo](http://docs.edi.kontur.ru/en/latest/methods/GetPartyInfo.html)
* [GetUsersInfo](http://docs.edi.kontur.ru/en/latest/methods/GetUsersInfo.html)
* [SendMessage](http://docs.edi.kontur.ru/en/latest/methods/SendMessage.html)

**Authenticate**

This is the method for receiving a security token.

Resource name: **/V1/Authenticate**

HTTP method: **POST**

HTTP header Authorization with the parameters konturediauth\_api\_client\_id, konturediauth\_login и konturediauth\_password (see Authorization) must be present in the request.

The server response will be UTF-8-coded string – a security token.

**GetAccessiblePartiesInfo**

This is the method for receiving the list of organizations available for a user.

Resource name: **/V1/Parties/GetAccessiblePartiesInfo**

HTTP method: **GET**

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return PartiesInfo structure.

**GetBoxDocumentsSettings**

This method provides documents settings for a box as well as the information on which documents and to which counterparties can be sent via the system and which messages can be received.

Resource name: **/V1/Messages/GetBoxDocumentsSettings**

HTTP method: **GET**

Query string parameters:

* boxld – the box identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return BoxDocumentsSettings structure.

**GetBoxesInfo**

This is the method for receiving the list of boxes available for a user.

Resource name: **/V1/Boxes/GetBoxesInfo**

HTTP method: **GET**

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return BoxesInfo structure.

**GetEvents**

This is the method for receiving all events in a box.

Resource name: **/V1/Messages/GetEvents**

HTTP method: **GET**

Query string parameters:

* boxld – the box identifier;
* exclusiveEventId – the identifier of the last received event. Must be empty if no one event has been found. The method will return events performed only after the event with the identifier exclusiveEventId;
* count – the number of the events requested. Optional. It must be between 1 to 1000. The server will return count events or less. If it has not been transferred, it is equal to 100 by default.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return BoxEventBatch structure. Present in the response LastEventId field value is to be used for the next method call.

**GetEventsFrom**

This is the method for receiving events in the box preformed after a certain moment of time.

Resource name: **/V1/Messages/GetEventsFrom**

HTTP method: **GET**

Query string paratemers:

* boxId – the box identifier;
* fromDateTime – the date and time string in the following format: YYYY-MM-DDThh:mm:ss.ss±hh:mm or YYYY-MM-DD. If date and time are transferred as a parameter, it is required to use “%2B” or “%2D” instead of “+” or “-“ respectively, for example “2016-04-06T14:30:00.00%2B05:00”. See more detailed information on data and time format here: <https://ru.wikipedia.org/wiki/ISO_8601>. The method will return events performed exactly after the time set in fromDateTime;
* count – the number of the events requested. Optional. It must be between 1 to 1000. The server will return count events or less. If it has not been transferred, it is equal to 100 by default.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return BoxEventBatch structure. Present in the response LastEventId field and GetEvents methods are to be used for the next method call.

**GetInboxMessage**

This is the method for receiving inbound message content.

Resource name: **/V1/Messages/GetInboxMessage**

HTTP method: **GET**

Query string parameters:

* boxId – the box identifier;
* messageId – the message identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return InboxMessageEntity structure.

**GetMainApiBox**

This is the method for receiving the box identifier. It is only used if the main kind of transport is Api. If it is not Api, the server will return 404 HTTP code (Not Found).

Resource name: **/V1/Boxes/GetMainApiBox**

HTTP method: **GET**

Query string parameters:

* partyId – the organization identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return BoxInfo structure.

**GetOrganizationCatalogueInfo**

This is the method for receiving the data on an organization structure.

Resource name: **/V1/Organizations/GetOrganizationCatalogueInfo**

HTTP method: **GET**

Query string parameters:

* partyId – the organization identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return OrganizationCatalogueInfo structure.

**GetOutboxMessage**

This is the method for receiving outbound message content.

Resource name: **/V1/Messages/GetOutboxMessage**

HTTP method: **GET**

Query string parameters:

* boxId – the box identifier;
* messageId – the message identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return OutboxMessageEntity structure.

**GetPartyInfo**

This is the method for receiving organization banking details. If a user does not have access to the organization, the server will return 403 HTTP code (Forbidden).

Resource name: **/V1/Parties/GetPartyInfo**

HTTP method: **GET**

Query string parameters:

* partyId – the organization identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return PartyInfo structure.

**GetUsersInfo**

This is the method for receiving the list of users having access to an organization.

Resource name: **/V1/Users/GetUsersInfo**

HTTP method: **GET**

Query string parameters:

* partyId – the organization identifier.

HTTP header Authorization with the data required for authentication must be present in the request. The server response will return UsersInfo structure.

**SendMessage**

This is the method for sending messages.

Resource name: **/V1/Messages/SendMessage**

HTTP method: **POST**

Query string parameters:

* boxId – the box identifier.

HTTP header Authorization with the data required for authentication must be present in the request. A byte array – the message content – must be sent in the request body. The server response will return OutboxMesageMeta structure which identifies an outbound message.

**Data structures**

To describe the structures we use the term *class.* Class is a combination of other classes or primitive types. The primitive types are the following:

* enum – enumeration: it is a fixed number of values serializing as a string written equal to one of the values,
* int – an integer,
* string – a string,
* dateTime – date and time serializing into a string according to ISO 8601 format (e.g. 2008-04-12T12:53Z),
* byte[] – a byte array serializing into the base64 string,
* bool – bool type.

**Data structures**

* [AcceptedRevocationInfo](http://docs.edi.kontur.ru/en/latest/structures/AcceptedRevocationInfo.html)
* [BankAccount](http://docs.edi.kontur.ru/en/latest/structures/BankAccount.html)
* [BasicMessageMeta](http://docs.edi.kontur.ru/en/latest/structures/BasicMessageMeta.html)
* [BoxDocumentSettings](http://docs.edi.kontur.ru/en/latest/structures/BoxDocumentSettings.html)
* [BoxesInfo](http://docs.edi.kontur.ru/en/latest/structures/BoxesInfo.html)
* [BoxEvent](http://docs.edi.kontur.ru/en/latest/structures/BoxEvent.html)
* [BoxEventBatch](http://docs.edi.kontur.ru/en/latest/structures/BoxEventBatch.html)
* [BoxEventContent](http://docs.edi.kontur.ru/en/latest/structures/BoxEventContent.html)
  + [MessageCheckingFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageCheckingFailEventContent.html)
  + [MessageCheckingOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageCheckingOkEventContent.html)
  + [MessageDeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDeliveredEventContent.html)
  + [MessageDiadocRevocationAcceptedEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDiadocRevocationAcceptedEventContent.html)
  + [MessageDiadocRevocationAcceptedForBuyerEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDiadocRevocationAcceptedForBuyerEventContent.html)
  + [MessageDocumentPackageSignedByMeFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByMeFailEventContent.html)
  + [MessageDocumentPackageSignedByMeOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByMeOkEventContent.html)
  + [MessageDocumentPackageSignedByRecipientFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByRecipientFailEventContent.html)
  + [MessageDocumentPackageSignedByRecipientOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByRecipientOkEventContent.html)
  + [MessageDraftOfDocumentPackageDeletedFromDiadocEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageDeletedFromDiadocEventContent.html)
  + [MessageDraftOfDocumentPackagePostedIntoDiadocEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackagePostedIntoDiadocEventContent.html)
  + [MessageDraftOfDocumentPackageSignedByMeEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageSignedByMeEventContent.html)
  + [MessageDraftOfDocumentPackageSignedBySenderEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageSignedBySenderEventContent.html)
  + [MessageReadByPartnerEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageReadByPartnerEventContent.html)
  + [MessageReceivedDiadocRoamingErrorEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageReceivedDiadocRoamingErrorEventContent.html)
  + [MessageUndeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageUndeliveredEventContent.html)
  + [NewInboxMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/NewInboxMessageEventContent.html)
  + [NewOutboxMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/NewOutboxMessageEventContent.html)
  + [RecognizeMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/RecognizeMessageEventContent.html)
  + [AmendmentRequestedEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/AmendmentRequestedEventContent.html)
  + [DiadocDocumentDeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/DiadocDocumentDeliveredEventContent.html)
  + [ProcessingTimesReportEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/ProcessingTimesReportEventContent.html)
* [BoxInfo](http://docs.edi.kontur.ru/en/latest/structures/BoxInfo.html)
* [BoxSettings](http://docs.edi.kontur.ru/en/latest/structures/BoxSettings.html)
* [ContactInformation](http://docs.edi.kontur.ru/en/latest/structures/ContactInformation.html)
* [DiadocUrls](http://docs.edi.kontur.ru/en/latest/structures/DiadocUrls.html)
* [DocumentDetails](http://docs.edi.kontur.ru/en/latest/structures/DocumentDetails.html)
* [DocumentId](http://docs.edi.kontur.ru/en/latest/structures/DocumentId.html)
* [DocumentSettings](http://docs.edi.kontur.ru/en/latest/structures/DocumentSettings.html)
* [DocumentSettingsForPartner](http://docs.edi.kontur.ru/en/latest/structures/DocumentSettingsForPartner.html)
* [ForeignAddressInfo](http://docs.edi.kontur.ru/en/latest/structures/ForeignAddressInfo.html)
* [GeoCoordinates](http://docs.edi.kontur.ru/en/latest/structures/GeoCoordinates.html)
* [InboxMessageEntity](http://docs.edi.kontur.ru/en/latest/structures/InboxMessageEntity.html)
* [InboxMessageMeta](http://docs.edi.kontur.ru/en/latest/structures/InboxMessageMeta.html)
* [IPInfo](http://docs.edi.kontur.ru/en/latest/structures/IPInfo.html)
* [MessageData](http://docs.edi.kontur.ru/en/latest/structures/MessageData.html)
* [Organization](http://docs.edi.kontur.ru/en/latest/structures/Organization.html)
* [OrganizationCatalogueInfo](http://docs.edi.kontur.ru/en/latest/structures/OrganizationCatalogueInfo.html)
* [OrganizationInfo](http://docs.edi.kontur.ru/en/latest/structures/OrganizationInfo.html)
* [OutboxMessageEntity](http://docs.edi.kontur.ru/en/latest/structures/OutboxMessageEntity.html)
* [Parties](http://docs.edi.kontur.ru/en/latest/structures/Parties.html)
* [PartiesInfo](http://docs.edi.kontur.ru/en/latest/structures/PartiesInfo.html)
* [Partner](http://docs.edi.kontur.ru/en/latest/structures/Partner.html)
* [PartyAddress](http://docs.edi.kontur.ru/en/latest/structures/PartyAddress.html)
* [PartyInfo](http://docs.edi.kontur.ru/en/latest/structures/PartyInfo.html)
* [ProcessingTimes](http://docs.edi.kontur.ru/en/latest/structures/ProcessingTimes.html)
* [RussianAddressInfo](http://docs.edi.kontur.ru/en/latest/structures/RussianAddressInfo.html)
* [RussianPartyInfo](http://docs.edi.kontur.ru/en/latest/structures/RussianPartyInfo.html)
* [ULInfo](http://docs.edi.kontur.ru/en/latest/structures/ULInfo.html)
* [UserInfo](http://docs.edi.kontur.ru/en/latest/structures/UserInfo.html)
* [UsersInfo](http://docs.edi.kontur.ru/en/latest/structures/UsersInfo.html)

**AcceptedRevocationInfo**

**class** **AcceptedRevocationInfo** {

DiadocDocumentType: DiadocDocumentType,

RevocationReason: string

}

The information on document revocation:

* DiadocDocumentType – a kind of a cancelled document,
* RevocationReason – a revocation cause.

**BankAccount**

**class** **BankAccount** {

BankAccountNumber: string,

CorrespondentAccountNumber: string,

BankId: string,

BankName: string

}

Banking details:

* BankAccountNumber – bank account number,
* CorrespondentAccountNumber – correspondent account,
* BankId – bank identifier code,
* BankName – bank name.

**BasicMessageMeta**

**class** **BasicMessageMeta** {

BoxId: string,

MessageId: string,

DocumentCirculationId: string

}

Message metainformation:

* BoxId – the identifier of a client’s box which received a message or where the message has been sent from,
* MessageId – the message identifier,
* DocumentCirculationId – the internal identifier of a message which allows to see the information on the message in the monitoring service.

**BoxDocumentsSettings**

**class** **BoxDocumentsSettings** {

BoxId: string,

DocumentsSettingsForPartner: DocumentSettingsForPartner[]

}

The settings that allow to exchange documents with different counterparties within one box:

* BoxId – the box identifier,
* DocumentsSettingsForPartner – documents settings for counterparties.

**BoxedInfo**

**class** **BoxesInfo** {

Boxes: BoxInfo[]

}

The information on boxes available for a user:

* BoxInfo[] – the information on all boxes available for a user.

**BoxEvent**

**class** **BoxEvent** {

BoxId: string,

PartyId: string,

EventId: string,

EventPointer: string,

EventDateTime: dateTime,

EventType: BoxEventType,

EventContent: BoxEventContent

}

The information on an event connected with an inbound or outbound message:

* BoxId – the box identifier,
* PartyId – the identifier of an organization which owns the box,
* EventId – the unique event identifier,
* EventPointer – the event pointer,
* EventDateTime – date and time of the event,
* EventType – a type of the event,
* EventContent – contents of an event.

**BoxEventBatch**

**class** **BoxEventBatch** {

Events: BoxEvent[],

LastEventId: string

}

The events set:

* Events – the list of events,
* LastEventId – the identifier of the last event in the set: during next GetEvents method call, it is the one which must be passed.

**BoxEventContent**

BoxEventContent holds the data on an event, may be one of the following types:

* [MessageCheckingFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageCheckingFailEventContent.html)
* [MessageCheckingOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageCheckingOkEventContent.html)
* [MessageDeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDeliveredEventContent.html)
* [MessageDiadocRevocationAcceptedEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDiadocRevocationAcceptedEventContent.html)
* [MessageDiadocRevocationAcceptedForBuyerEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDiadocRevocationAcceptedForBuyerEventContent.html)
* [MessageDocumentPackageSignedByMeFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByMeFailEventContent.html)
* [MessageDocumentPackageSignedByMeOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByMeOkEventContent.html)
* [MessageDocumentPackageSignedByRecipientFailEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByRecipientFailEventContent.html)
* [MessageDocumentPackageSignedByRecipientOkEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDocumentPackageSignedByRecipientOkEventContent.html)
* [MessageDraftOfDocumentPackageDeletedFromDiadocEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageDeletedFromDiadocEventContent.html)
* [MessageDraftOfDocumentPackagePostedIntoDiadocEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackagePostedIntoDiadocEventContent.html)
* [MessageDraftOfDocumentPackageSignedByMeEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageSignedByMeEventContent.html)
* [MessageDraftOfDocumentPackageSignedBySenderEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageDraftOfDocumentPackageSignedBySenderEventContent.html)
* [MessageReadByPartnerEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageReadByPartnerEventContent.html)
* [MessageReceivedDiadocRoamingErrorEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageReceivedDiadocRoamingErrorEventContent.html)
* [MessageUndeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/MessageUndeliveredEventContent.html)
* [NewInboxMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/NewInboxMessageEventContent.html)
* [NewOutboxMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/NewOutboxMessageEventContent.html)
* [RecognizeMessageEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/RecognizeMessageEventContent.html)
* [AmendmentRequestedEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/AmendmentRequestedEventContent.html)
* [DiadocDocumentDeliveredEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/DiadocDocumentDeliveredEventContent.html)
* [ProcessingTimesReportEventContent](http://docs.edi.kontur.ru/en/latest/structures/events/ProcessingTimesReportEventContent.html)

**MessageCheckingFailEventContent**

**class** **MessageCheckingFailEventContent** {

OutboxMessageMeta: BasicMessageMeta,

Errors: string[],

ReportNumber: string

}

The information on a new event – an outbound message has not passed the check on the recipient’s side. It matches BoxEventType = MessageCheckingFail.

* OutboxMessageMeta – metainformation of the checked message,
* Errors – errors found in the message which has not passed the check,
* ReportNumber – status message number corresponding to the given event: relevant only for status messages sent according to the scheme of forming documents after checking Invoic message.

**MessageCheckingOkEventContent**

**class** **MessageCheckingOkEventContent** {

OutboxMessageMeta: BasicMessageMeta,

Warnings: string[],

ReportNumber: string

}

The information on a new event – an outbound message passed the check on the recipient’s side. Matches BoxEventType = MessageCheckingOk.

* OutboxMessageMeta – checked message meatinformation,
* Warnings – notes for the message which has passed the check,
* ReportNumber – status message number corresponding to the given event: relevant only for status messages sent according to the scheme of forming documents after checking Invoic message.

**MessageDeliveredEventContent**

**class** **MessageDeliveredEventContent** {

OutboxMessageMeta: BasicMessageMeta

}

The information on a new event – an outbound message has been delivered. Matches BoxEventType=MessageDelivered.

* OutboxMessageMeta – metainformation of a delivered message.

**MessageDiadocRevocationAcceptedEventContent**

**class** **MessageDiadocRevocationAcceptedEventContent** {

AcceptedRevocationInfo: AcceptedRevocationInfo,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

OutboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a sender’s box – the document based on the outbound Invoic message has been cancelled in Diadoc. Matches BoxEventType = DiadocRevocationAccepted.

* AcceptedRevocationInfo – the information on a cancelled document,
* DiadocBoxId – the identifier of a box in Diadoc whereof signed documents have been sent by a sender,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* OutboxMessageMeta – metainformation of an outbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDiadocRevocationAcceptedForBuyerEventContent**

**class** **MessageDiadocRevocationAcceptedForBuyerEventContent** {

AcceptedRevocationInfo: AcceptedRevocationInfo,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

DiadocUrls: DiadocUrls,

InboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a recipient’s box – the document based on the inbound Invoic message has been cancelled. Matches BoxEventType = DiadocRevocationAcceptedForBuyer.

* AcceptedRevocationInfo – the information on a cancelled document,
* DiadocBoxId – the identifier of a box in Diadoc in which documents signed by a sender have been delivered,
* InvoiceId – the received VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the received TORG-12 delivery note identifier,
* InvoiceCorrectionId – the received adjustment invoice identifier,
* UniversalTransferDocumentId – the received UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the received UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* InboxMessageMeta – metainformation of an inbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDocumentPackageSignedByMeFailEventContent**

**class** **MessageDocumentPackageSignedByMeFailEventContent** {

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

InboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a recipient’s box – the documents based on the inbound Invoic message and correction documents based on the inbound Coinvoic message have been rejected. Matches BoxEventType = DocumentPackageSignedByMeFail.

* DiadocBoxId – the identifier of a box in Diadoc in which documents signed by a sender have been delivered,
* InvoiceId – the received VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the received TORG-12 delivery note identifier,
* InvoiceCorrectionId – the received adjustment invoice identifier,
* UniversalTransferDocumentId – the received UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the received UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* InboxMessageMeta – metainformation of an inbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDocumentPackageSignedByMeOkEventContent**

**class** **MessageDocumentPackageSignedByMeOkEventContent** {

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

InboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a recipient’s box – the documents based on the inbound Invoic message and correction documents based on the inbound Coinvoic message have been signed. Matches BoxEventType = DocumentPackageSignedByRecipientOk.

* DiadocBoxId – the identifier of a box in Diadoc in which documents signed by a sender have been delivered,
* InvoiceId – the received VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the received TORG-12 delivery note identifier,
* InvoiceCorrectionId – the received adjustment invoice identifier,
* UniversalTransferDocumentId – the received UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the received UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* InboxMessageMeta – metainformation of an inbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDocumentPackageSignedByRecipientFailEventContent**

**class** **MessageDocumentPackageSignedByRecipientFailEventContent** {

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

OutboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a sender’s box – the documents based on the outbound Invoic message and correction documents based on the outbound Coinvoic have been rejected. Matches BoxEventType = DocumentPackageSignedByRecipientFail.

* DiadocBoxId – the identifier of a box in Diadoc whereof signed documents have been sent,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* OutboxMessageMeta – metainformation of an outbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDocumentPackageSignedByRecipientOkEventContent**

**class** **MessageDocumentPackageSignedByRecipientOkEventContent** {

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

OutboxMessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a sender’s box – a TORG-12 delivery note, a VAT invoice or a UTD based on the outbound Invoic message as well as correction documents based on the outbound Coinvoic message have been signed. Matches BoxEventType = DocumentPackageSignedByRecipientOk.

* DiadocBoxId – the identifier of a box in Diadoc whereof signed documents have been sent,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* OutboxMessageMeta – metainformation of an outbound Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDraftOfDocumentPackageDeletedFromDiadocEventContent**

**class** **MessageDraftOfDocumentPackageDeletedFromDiadocEventContent** {

OutboxMessageMeta: BasicMessageMeta,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

DiadocUrls: DiadocUrls,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event – documents drafts based on the outbound Invoic message have been deleted. Matches BoxEventType = DraftOfDocumentPackageDeletedFromDiadoc.

* OutboxMessageMeta – metainformation of a sent Invoic message,
* DiadocBoxId – the identifier of a box in Diadoc whereof documents drafts have been deleted,
* InvoiceId – the deleted VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the deleted TORG-12 delivery note identifier,
* InvoiceCorrectionId – the deleted adjustment invoice identifier,
* UniversalTransferDocumentId – the deleted UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the deleted UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDraftOfDocumentPackagePostedIntoDiadocEventContent**

**class** **MessageDraftOfDocumentPackagePostedIntoDiadocEventContent** {

OutboxMessageMeta: BasicMessageMeta,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event – Diadoc documents drafts based on the outbound Invoic message have been formed. Matches BoxEventType = DraftOfDocumentPackagePostedIntoDiadoc.

* OutboxMessageMeta – metainformation of a sent Invoic message,
* DiadocBoxId – the identifier of a box in Diadoc in which documents drafts have been formed,
* InvoiceId – the identifier of a VAT invoice draft,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the identifier of a TORG-12 delivery note draft,
* InvoiceCorrectionId – the identifier of an adjustment invoice draft,
* UniversalTransferDocumentId – the identifier of a UTD (Universal Transfer Document) draft,
* UniversalCorrectionDocumentId – the identifier of a UCD (Universal Correction Document) draft,
* DiadocUrls – the links to documents in Diadoc,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDraftOfDocumentPackageSignedByMeEventContent**

**class**

**MessageDraftOfDocumentPackageSignedByMeEventContent** {

OutboxMessageMeta: BasicMessageMeta,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

DiadocUrls: DiadocUrls,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a sender’s box – documents drafts based on the outbound Invoic message have been signed and sent. Matches BoxEventType = DraftOfDocumentPackageSignedByMe.

* OutboxMessageMeta – metainformation of a sent Invoic message,
* DiadocBoxId – the identifier of a box in Diadoc whereof signed documents drafts have been sent,
* InvoiceId – the signed VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the signed TORG-12 delivery note identifier,
* InvoiceCorrectionId – the received adjustment invoice identifier,
* UniversalTransferDocumentId – the received UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the received UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageDraftOfDocumentPackageSignedBySenderEventContent**

**class** **MessageDraftOfDocumentPackageSignedBySenderEventContent** {

InboxMessageMeta: BasicMessageMeta,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

DiadocUrls: DiadocUrls,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a recipient’s box – documents drafts based on the inbound Invoic message have been signed and sent. Matches BoxEventType = DraftOfDocumentPackageSignedBySender.

* InboxMessageMeta – metainformation of a received Invoic message,
* DiadocBoxId – the identifier of a box in Diadoc in which documents signed by a sender have been delivered,
* InvoiceId – the received VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the received TORG-12 delivery note identifier,
* InvoiceCorrectionId – the received adjustment invoice identifier,
* UniversalTransferDocumentId – the received UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the received UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageReadByPartnerEventContent**

**class** **MessageReadByPartnerEventContent** {

OutboxMessageMeta: BasicMessageMeta

}

The information on a new event – an outbound message has been read. Matches BoxEventType= MessageReadByPatner.

* OutboxMessageMeta – metainformation of a read message.

**MessageReceivedDiadocRoamingErrorEventContent**

**class** **MessageReceivedDiadocRoamingErrorEventContent** {

OutboxMessageMeta: BasicMessageMeta,

Reason: string,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

DiadocUrls: DiadocUrls,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event – errors in the documents based on the outbound Invoic message have been found. Matches BoxEventType = ReceivedDiadocRoamingError.

* OutboxMessageMeta – metainformation of an outbound Invoic message,
* Reason – the errors found in the documents which have not passed the check on the recipient’s side,
* DiadocBoxId – the identifier of a box in Diadoc whereof documents signed by a sender have been sent,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* UnversalDocumentFunction – UTD function of an Invoic message.

**MessageUndeliveredEventContent**

**class** **MessageUndeliveredEventContent** {

OutboxMessageMeta: BasicMessageMeta,

MessageUndeliveryReasons: string[]

}

The information on a new event – delivery error of an outbound message. Matches BoxEventType = MessageUndelivered.

* OutboxMessageMeta – metainformation of an undelivered message,
* MesageUndeliveryReasons – description of the reasons by which the message has been undelivered.

**NewInboxMessageEventContent**

**class** **NewInboxMessageEventContent** {

InboxMessageMeta: InboxMessageMeta

}

The information on a new event – an inbound message has been received. Matches BoxEventType = NewInboxMessage.

* InboxMesageMeta – metainformation of the message.

**NewOutboxMessageEventContent**

**class** **NewOutboxMessageEventContent** {

OutboxMessageMeta: BasicMessageMeta

}

The information on a new event – an outbound message has been sent. Matches BoxEventType = NewOutboxMessage.

* OutboxMessageMeta – metainformation of the message.

**RecognizeMessageEventContent**

**class** **RecognizeMessageEventContent** {

OutboxMessageMeta: BasicMessageMeta,

DocumentType: DocumentType,

SenderPartyId: string,

RecipientPartyId: string

}

The information on a new event – an outbound message has been parsed and its main parameters (format, type, sender and recipient) have been successfully identified. Matches BoxEventType = RecognizeMessage.

* OutboxMessageMeta – metainformation of the message.
* DocumentType – the type of the message.
* SenderPartyId – a sender of the message.
* RecipientPartyId – a recipient of the message.

**AmendmentRequestedEventContent**

**class** **AmendmentRequestedEventContent** {

AmendmentRequestMessage: string,

DiadocBoxId: string,

InvoiceId: string,

MessageId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

MessageMeta: BasicMessageMeta,

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a box – a note on amendment of a document based on the outbound Invoic message has been received. Matches BoxEventType = AmendmentRequqsted.

* AmendmentRequestMessage – the text of the note on amendment,
* DiadocBoxId – the identifier of a box in Diadoc whereof documents signed by a sender have been sent,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* MessageMeta – metainformation of the Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**DiadocDocumenttDeliveredEventContent**

**class** **DiadocDocumentDeliveredEventContent** {

DiadocBoxId: string,

MessageId: string,

InvoiceId: string,

Torg12Id: string,

InvoiceCorrectionId: string,

UniversalTransferDocumentId: string,

UniversalCorrectionDocumentId: string,

DiadocUrls: DiadocUrls,

OutboxMessageMeta: BasicMessageMeta

UniversalDocumentFunction: UniversalDocumentFunctionType

}

The information on a new event in a box – a note on receiving the document based on the outbound Invoic message has been delivered. Matches BoxEventType = DiadocDocumentDelivered.

* DiadocBoxId – the identifier of a box in Diadoc whereof documents signed by a sender have been sent,
* InvoiceId – the sent VAT invoice identifier,
* MessageId – the message identifier in Diadoc,
* Torg12Id – the sent TORG-12 delivery note identifier,
* InvoiceCorrectionId – the sent adjustment invoice identifier,
* UniversalTransferDocumentId – the sent UTD (Universal Transfer Document) identifier,
* UniversalCorrectionDocumentId – the sent UCD (Universal Correction Document) identifier,
* DiadocUrls – the links to documents in Diadoc,
* OutboxMessageMeta – metainformation of the Invoic message,
* UnversalDocumentFunction – UTD function of an Invoic message.

**ProcessingTimesReportEventContent**

**class** **ProcessingTimesReportEventContent** {

DocumentCirculationId: string,

SenderPartyId: string,

RecipientPartyId: string,

InitialMessageId: string,

InitialDocumentId: DocumentId,

DocumentType: DocumentType,

DocumentCirculationStartTimestamp: DateTime,

DocumentCirculationCompletionTimestamp: DateTime,

ProcessingTimes: ProcessingTimes

}

The information on a new event – an outbound message has been parsed and its main parameters (format, type, sender and recipient) have been successfully identified. Matches BoxEventType = ProcessingTimesReport.

* DoucmentCirculationId – the internal message identifier, this identifier allows to see the information on a message in the monitoring service.
* SenderPartyId – the identifier of a sender’s organization,
* RecipientPartyId – the identifier of a recipient’s organization.
* InitialMessageId – the message identifier,
* InitialDocumentId – the document identifier in a box,
* DocumentType – the type of the message,
* DocumentCirculationStartTimestamp – the time of the document exchange beginning,
* DocumentCirculationCompletionTimestamp – the time of the document exchange completion,
* ProcessingTimes – the time spent on different stages of the message processing and total time of message delivery.

**BoxInfo**

**class** **BoxInfo** {

Id: string,

PartyId: string,

Gln: string,

IsTest: bool,

BoxSettings: BoxSettings

}

The information on abox:

* Id – the box identifier,
* PartyId – the identifier of an organization which owns the box,
* Gln – the gln of the box,
* IsTest – the flag indicating that a box is under test,
* BoxSettings – box settings.

**BoxSettings**

**class** **BoxSettings** {

TransportType: TransportType,

IsMain: bool,

DocumentTypes: DocumentType,

CustomMessageFormats: MessageFormat,

InboxRelativePath: string,

OutboxRelativePath: string,

ProviderTransportBoxId: string

}

Box settings:

* TransportType – the kind of transport used in a box,
* IsMain – traits of the main transport,
* DoucmentTypes – kinds of messages set for a box, “Any” value means that the box corresponds to the “Default transport” settings,
* CustomMessageFormats – message formats set for a box, “Any” value means that the box corresponds to the “Default transport” settings,
* InboxRelativePath – a folder path of inbound messages to Ftp: it shall be completed only if the transport type is “Ftp” and the box is used for inbound messages,
* OutboxRelativePath – a folder path of outbound messages to Ftp: it shall be completed only if the transport type is “Ftp” and the box is used for outbound messages,
* ProviderTransportBoxId – the identifier of a provider’s box: it shall be completed only if the transport type is “Provider”.

**ContactInformation**

**class** **ContactInformation** {

Name: string,

Phone: string,

Fax: string,

Email: string

}

Contact information:

* Name – full name,
* Phone – phone number,
* Fax – fax,
* Email – email.

**DiadocUrls**

**class** **DiadocUrls** {

MessageUrl: string,

InvoiceUrl: string,

Torg12Url: string,

InvoiceCorrectionUrl: string,

UniversalTransferDocumentUrl: string,

UniversalCorrectionDocumentUrl: string,

NonformalizedUrl: string

}

The lonks to documents in Diadoc:

* MessageUrl – the link to a message in Diadoc,
* InvoiceUrl – the link to a VAT invoice in Diadoc,
* Torg12Url – the link to a TORG-12 delivery note in Diadoc,
* InvoiceCorrectionUrl – the link to an adjustment invoice in Diadoc,
* UniversalTransferDocumentUrl – the link to a UTD in Diadoc,
* UniversalCorrectionDocumentUrl – the link to a UCD in Diadoc,
* NonformalizedUrl – the link to a nonformalized document in Diadoc.

**DocumentDetails**

**class** **DocumentDetails** {

DocumentType: DocumentType,

DocumentIsTest: bool,

DocumentNumber: string,

DocumentDate: dateTime

}

The information on a document:

* DocumentType – the type of a document,
* DocumentTest – the flag indicating that the document is under test,
* DocumentNumber – the number of a document,
* DocumentDate – the date of a document.

**DocumentId**

**class** **DocumentId** {

BoxId: string,

EntityId: string,

}

The identifier of a document in the box:

* BoxId – the identifier of an organization box,
* EntityId – the document entity identifier in the organization box.

**DocumentSettings**

**class** **DocumentSettings** {

DocumentType: DocumentType,

DocumentDirection: DocumentDirection

}

The atomic setting for sending/receiving documents of a certain type:

* DocumentType – the type of a document,
* DocumentDirection – the direction of a document.

**DocumentSettingsForPartner**

**class** **DocumentSettingsForPartner** {

Partner: Partner,

DocumentSettings: DocumentSettings[]

}

The settings for document exchange with a certain counterparty:

* Partner – a counterparty organization,
* DocumentSettings – a set of documents for exchanging.

**ForeignAddressInfo**

**class** **ForeignAddressInfo** {

CountryCode: string,

Address: string

}

A foreign address:

* CountryCode – two-letter country code according to ISO 3166,
* Address – address.

**GeoCoordinates**

**class** **GeoCoordinates** {

Latitude: string,

Longitude: string

}

Geographical coordinates:

* Latitude – latitude,
* Longitude – longitude.

**InboxMessageEntity**

**class** **InboxMessageEntity** {

Meta: InboxMessageMeta,

Data: MessageData

}

An inbound message:

* Meta – message metainformation,
* Data – message data.

**InboxMessageMeta**

**class** **InboxMessageMeta** {

BoxId: string,

MessageId: string,

DocumentCirculationId: string,

SendDateTime: dateTime,

Sender: Partner,

MessageFormat: MessageFormat,

DocumentDetails: DocumentDetails

}

An inbound message metainformation:

* BoxId – the identifier of a box in which the message has been delivered,
* MessageId – the message identifier,
* DocumentCirculationId – the internal message identifier: this identifier allows to see the information on the message in the monitoring service,
* SendDateTime – date and time of the message sent by a counterparty,
* Sender – an organization-sender of a message,
* MessageFormat – message format,
* DoumentDetails – the infomation on a document.

**IPInfo**

**class** **IPInfo** {

Inn: string,

FirstName: string,

MiddleName: string,

LastName: string

}

An individual’s details:

* Inn – tax payer ID,
* FirstName – first name,
* MiddleName – middle name,
* LastName – last name.

**MessageData**

**class** **MessageData** {

MessageFileName: string,

MessageBody: byte[]

}

Message content:

* MessageLikeName – file name of a message used while sending the message by special transports (e.g. FTP),
* MesageBody – message body.

**Organization**

**class** **Organization** {

OrganizationDate: dateTime,

OrganizationInfo: OrganizationInfo

}

The information on an organization or a delivery/dispatch point:

* OrganizationDate – the beginning of work date,
* OrganizationInfo – the information on an organization or a delivery/dispatch point.

**OrganizationCatalogueInfo**

**class** **OrganizationCatalogueInfo** {

Organizations: Organization[],

DeliveryPoints: Organization[]

}

The information on an organization structure:

* Organizations – related legal entities and/or entrepreneurs,
* DeliveryPoints – delivery/dispatch points.

**OrganizationInfo**

**class** **OrganizationInfo** {

Gln: string,

PartyAddress: PartyAddress,

GeoCoordinates: GeoCoordinates,

RussianPartyInfo: RussianPartyInfo,

BankAccount: BankAccount,

SupplierCodeInBuyerSystem: string,

BusinessEntityMercuryId: string,

Chief: ContactInformation,

Bookkeeper: ContactInformation,

SalesAdministration: ContactInformation,

OrderContact: ContactInformation,

LocalizationType: string

}

Banking details and address of an organization or delivery/dispatch points:

* Gln – Gln,
* PartyAddress – address,
* GeoCoordinates – geographical coordinates,
* RussianPartyInfo – company details,
* BankAccount – banking details,
* SupplerCodeInBuyerSystem – the additional identifier,
* BusinessEntityMercuryId – the business entity identifier in the Federal State Informational System “Merkuriy”,
* Chief – data on a director of an organzation,
* Bookkeeper – data on an accountant,
* SalesAdministration – data on a manager,
* OrderContact – data on a contact person,
* LocalizationType – type of an organization.

**OutboxMessageEntity**

**class** **OutboxMessageEntity** {

Meta: OutboxMessageMeta,

Data: MessageData

}

An outbound message:

* Meta – metainformation of an outbound message,
* Data – an outbound message data.

**Parties**

**class** **Parties** {

PartyInfo: PartyInfo[]

}

The list of organizations:

* PartyInfo – organization details.

**PartiesInfo**

**class** **PartiesInfo** {

Parties:Parties

}

The information on organizations:

* Parties – the list of organizations.

**Partner**

**class** **Partner** {

PartnerId: string,

PartnerGln: string,

PartnerName: string

}

The data on an organization registered in the system (used to transfer information on counterparties):

* PartnerId – the unique identifier of an organization in the system,
* PartnerGln – GLN,
* PartnerName – name.

**PartyAddress**

**class** **PartyAddress** {

RussianAddressInfo: RussianAddressInfo,

ForeignAddressInfo: ForeignAddressInfo

}

Organization address or a delivery/dispatch point:

* RussianAddressInfo – Russian address,
* ForeignAddressInfo – foreign address.

**PartyInfo**

**class** **PartyInfo** {

Id: string,

Gln: string,

Name: string,

Inn: string,

Kpp: string,

PartyTypeCode: string,

DiadocOrgId: string,

OrganizationCatalogueUpdateTime: dateTime

}

Organization details:

* Id – the organization identifier,
* Gln – organization GLN,
* Name – name,
* Inn – tax payer ID,
* Kpp – registration reason code,
* PartyTypeCode – type of an organization,
* DiadocOrgId – the identifier in Diadoc,
* OrganizationCatalogueUpdateTime – time of the last organization details or its structure changing.

**ProcessingTimes**

**class** **ProcessingTimes** {

TotalWorkTime: TimeSpan,

DeliveryTime: TimeSpan,

ConnectorWorkTime: TimeSpan,

ConnectorWaitTime: TimeSpan,

DiadocWaitTime: TimeSpan,

RecadvWaitTime: TimeSpan,

DeliveryNotificationWaitTime: TimeSpan

}

The time spent on different stages of message processing and total time of message delivery:

* TotalWorkTime – the total time of message delivery which is the gap between DocumentCirculationCompletiontimeptamp and DocumentCirculationStartTimestamp,
* DeliveryTime – the delivery time excluding the time of waiting for custom actions or custom integration solutions.
* ConnectorWorkTime – the time of message processing by the Connector,
* ConnectorWaitTime – the wait time of conditions fulfillment required to continue processings by the Connector.
* DiadocWaitTime – the time between forming documents drafts in Diadoc and their signing,
* RecadvWaitTime – the wait time of receiving advice while using correspondent interaction schemes between partners,
* DeliveryNotificationWaitTime – the time from the first attempt to deliver a message to the moment of receiving delivery confirmation.

**RussianAddressInfo**

**class** **RussianAddressInfo** {

PostalCode: string,

RegionCode: string,

District: string,

City: string,

Village: string,

Street: string,

House: string,

Flat: string

}

Russian address:

* PostalCode – postcode,
* RegionCode – region code in ISO format,
* District – district,
* City – city,
* Village – locality,
* Street – street,
* House – building,
* Flat – flat/office.

**RussianPartyInfo**

**class** **RussianPartyInfo** {

ULInfo: ULInfo,

IPInfo: IPInfo

}

Organization or delivery/dispatch points details:

* ULInfo – legal entity details,
* IPInfo –details of an individual.

**ULInfo**

**class** **ULInfo** {

Inn: string,

Kpp: string,

Name: string

}

Legal entity details:

* Inn – tax payer ID,
* Kpp - registration reason code,
* Name – name.

**UserInfo**

**class** **UserInfo** {

Email: string

}

The information on a user:

* Email – email.

**UsersInfo**

**class** **UsersInfo** {

Users: UserInfo[]

}

The information on the list of users:

* Users – the list if users.

**Enumerations**

**Enumerations**

* [BoxEventType](http://docs.edi.kontur.ru/en/latest/enums/BoxEventType.html)
* [DiadocDocumentType](http://docs.edi.kontur.ru/en/latest/enums/DiadocDocumentType.html)
* [DocumentDirection](http://docs.edi.kontur.ru/en/latest/enums/DocumentDirection.html)
* [DocumentType](http://docs.edi.kontur.ru/en/latest/enums/DocumentType.html)
* [MessageFormat](http://docs.edi.kontur.ru/en/latest/enums/MessageFormat.html)
* [PartyType](http://docs.edi.kontur.ru/en/latest/enums/PartyType.html)
* [TransportType](http://docs.edi.kontur.ru/en/latest/enums/TransportType.html)
* [UniversalDocumentFunctionType](http://docs.edi.kontur.ru/en/latest/enums/UniversalDocumentFunctionType.html)

**Types of events**

**enum** BoxEventType {

Unknown,

NewOutboxMessage,

NewInboxMessage,

RecognizeMessage,

MessageDelivered,

MessageUndelivered,

MessageReadByPartner,

MessageCheckingOk,

MessageCheckingFail,

DraftOfDocumentPackagePostedIntoDiadoc,

DraftOfDocumentPackageSignedByMe,

DraftOfDocumentPackageDeletedFromDiadoc,

DraftOfDocumentPackageSignedBySender,

ReceivedDiadocRoamingError,

DiadocRevocationAccepted,

DiadocRevocationAcceptedForBuyer,

DocumentPackageSignedByRecipientOk,

DocumentPackageSignedByMeOk,

DocumentPackageSignedByRecipientFail,

DocumentPackageSignedByMeFail,

AmendmentRequested,

ProcessingTimesReport,

DiadocDocumentDelivered

}

**Types of documents in Diadoc**

**enum** DiadocDocumentType {

Invoice,

Torg12,

CorrectiveInvoice,

UniversalTransferDocument,

UniversalCorrectionDocument

}

**Document directions**

**enum** DocumentDirection {

Unknown,

FromMe,

ToMe,

Bidirectional

}

**Types of documents**

**enum** DocumentType {

Unknown,

Any,

Orders,

Ordrsp,

Desadv,

Recadv,

Invoic,

Coinvoic,

Alcrpt,

Stsmsg,

Retann,

Retins,

Retdes,

Retrec,

Retinv,

POrders,

Pricat,

PriceList,

Partin,

Delfor,

Invrpt,

Slsrpt,

Iftmbf,

Iftmbc

}

**Unknown and Any are special values returning in the following cases:**

* Unknown – the type of message has not been identified.
* Any – a certain type of a message has not been specified. The value may return during calling GetBoxesInfo method in the BoxSettings structure.

**Messages formats**

**enum** MessageFormat {

Unknown,

Any,

KonturXml,

KorusXml,

Eancom2002,

EcrRusXml

}

**Unknown and Any are special values returning in the following cases:**

* Unknown – the format of a message has not been identified.
* Any – a certain format of a message has not been specified. The value may return during calling GetBoxesInfo method in the BoxSettings structure.

**Types of organizations**

**enum** PartyType {

Unknown,

Buyer,

Supplier,

Distributor

}

**Transport types used in boxes**

**enum** TransportType {

Unknown,

Api,

As2,

Ftp,

Provider

}

**Functions of the UTD-type documents**

**enum** UniversalDocumentFunctionType {

Invoice,

Basic,

InvoiceAndBasic

}

**API revision history**

24.04.2018

Regarding BoxEvent structure modifications, recommended APi access has been updated: if necessary to get new events list beginning with a certain event, it is required to use **EventPointer** in the exclusiveEventId parameter while sending GetEvents request. EventId identifier shall not be supported as a parameter value in the following updates.

Updated event and structure descriptions:

* EventPointer has been added into the BoxEvent,
* UTD (Universal Transfer Document) function has been added into event descriptions connected with documents in Diadoc,
* ProcessingTimesReportEvent and DiadocDocumentDeliveredEvent event descriptions have been added,
* Spelling mistakes and deficiencies have been corrected.

29.05.2017

API EDI.Kontur documentation has changes its interface. The new interface provides the following:

* Practical tree menu,
* Search system on pages,
* Modifications descriptions page.

If you have noticed any mistakes, you may offer their corrections by clicking the “Edit on GitHub” link on the correspondent page.