# SIDDIN 2.1
# DATA EXCHANGE

## SIDDIN 2.1 WEBSERVICE
## TECHNICAL SPECIFICATION

Version of the documentation 1.2 dated 2022-07-25

Document status

Internal development

Keywords

Economic Information Bureau, BIG, Tranche, WebService, SIDDIN, NICCI

KRD BIG S.A., Danuty Siedzikówny 12, 51-214 Wrocław
https://en.krd.pl

## Document attributes

| | Attribute | Value |
|---|---|---|
| | A | B |
| 1 | Project | Siddin 2.1 |
| 2 | Title | Data exchange |
| 3 | Subtitle | Siddin 2.1 WebService technical specification |
| 4 | Documentation version | 1.2 |
| 5 | Version time | 2022-07-25 |
| 6 | File | Siddin 2.1 - Technical specification.docx / pdf |
| 7 | Authors | Rafał Stramski |
| 8 | Supervision | Sebastian Tkocz |
| 9 | Copyright | Copyright © Krajowy Rejestr Długów, 2006-2022 |
| 10 | Comments | |

## History of the document

| Attribute | | Value | Date |
|---|---|---|---|
| A | | B | C |
| 1 | Documentation version | 1.0 | 2006-10-13 |
| 2 | Author | Rafał Stramski | |
| 3 | Content proofread by | | |
| 4 | Form proofread by | | |
| 5 | Approved by | | |
| 6 | Description | | |
| 1 | Documentation version | 1.1 | 2020-06-08 |
| 2 | Author | Maciej Łukasik | |
| 3 | Content proofread by | Maciej Łukasik | 2020-06-08 |
| 4 | Form proofread by | Maciej Łukasik | 2020-06-08 |
| 5 | Approved by | Maciej Łukasik | 2020-06-08 |
| 6 | Description | 1. Updating / refreshing the description of technical content. | |
| 1 | Documentation version | 1.2 | 2022-07-25 |
| 2 | Author | Maciej Łukasik | |
| 3 | Content proofread by | Maciej Łukasik | 2022-07-25 |
| 4 | Form proofread by | Maciej Łukasik | 2022-07-25 |
| 5 | Approved by | Maciej Łukasik | 2022-07-25 |
| 6 | Description | 1. Updating / refreshing the description of technical content.<br>2. Creating a new automatic table of contents.<br>3. Modification / addition of WSDL address content description.<br>4. Adding information about Nicci 3.2 protocol support (Introduction, NicciVersionEnum type). | |
| 1 | Documentation version | | |
| 2 | Author | | |
| 3 | Content proofread by | | |
| 4 | Form proofread by | | |
| 5 | Approved by | | |
| 6 | Description | | |
| 1 | Documentation version | | |
| 2 | Author | | |
| 3 | Content proofread by | | |
| 4 | Form proofread by | | |
| 5 | Approved by | | |
| 6 | Description | | |
| 1 | Documentation version | | |
| 2 | Author | | |
| 3 | Content proofread by | | |
| 4 | Form proofread by | | |
| 5 | Approved by | | |
| 6 | Description | | |

# Table of Contents

# Introduction

The KRD system provides the services of the Economic Information Bureau (BIG) pursuant to the Act on Disclosure of Economic Information and Exchange of Economic Data, dated 9 April, 2010. It enables adding, updating and deleting economic information and sharing it with third parties.

The economic information may be entered into the KRD system with one of the two available methods: interaction on the website of the bureau – KRD WWW Customer Panel (manually or via CSV files), or via Web services using the SOAP protocol. One of such Web services is called SIDDIN, version 2.1.

WebService SIDDIN 2.1 is intended for processing of the NICCI protocol tranches. It is enabling to process all published versions of the NICCI protocol. The NICCI protocol (version 2.1, 3.0, 3.1, 3.2) has been described in the separate technical specifications.

Version 2.1 of the service guarantees a possibility of an asynchronous processing of a great amount of data. The service supports intake of large XML data (consistent with the NICCI protocol) in "chunks" and its background processing. The user has also got the possibility to monitor the progress of works and download the resulting data in parts.

This document describes methods of the SIDDIN 2.1 service and different ways of connecting the Clients to SIDDIN 2.1 server. The SIDDIN 2.1 service will be hereinafter referred to as the SIDDIN (without giving the version's number). The methods described in this document do not regard the previous versions of the service.

**NOTE!**

**Please send us one aggregate tranche of Nicci / several aggregate tranches of Nicci at the maximum.**

Sending many individual tranches with an order, e.g. adding only one case, means that their waiting for processing / processing on our side takes longer (sometimes much longer) than the processing of one aggregate tranche with multiple orders, such as case additions.

That is why, the tutorial which you receive from us in the implementation process, shows how multiple cases and related obligations are added through one aggregate tranche. Update, removal, suspension, unsuspension and the like orders should be handled in a similar way. All orders may, and even should, be included in one aggregate tranche.

**Additionally, we ask for:**
- aggregation of negative obligations of the same debtor in one negative case instead of e.g. several negative cases,

- aggregation of positive obligations of the same contractor in one positive case instead of e.g. several positive cases.

# 1. SIDDIN service methods

## 1.1. General data

The SIDDIN service uses messages to perform methods and return their results. Incoming messages are suffixed with the word "Request". Results of the methods constitute of determined data structures or, in some cases – of a single data.

## 1.2. Authentification of the Client

SIDDIN server authenticates the Client by verifying the correctness of user's login and password. A correct self-authentication to the system results in providing the Client with a ticket allowing using the SIDDIN server for a determined period of time (24h).

In order to authenticate himself, the Client sets off the "Login" method with the "LoginRequest" parameter. The "LoginRequest" message has got two properties in a form of strings: user's login ("UserName") and password ("Password"). This method returns a string containing the obtained ticket (at least 40 characters) or no element data if the self-authentication failed. After finishing work with the server, it is required to log out (invalidate) the obtained ticket, providing "LogoutRequest" message with the current ticket as the parameter of the "Logout" method. If the logout does not take place, then the ticket is valid for a limited period of time (24h), and then will be automatically invalidated.

## 1.3. Data loading

The service supports loading of large XML data in smaller parts of unlimited quantity, with maintenance of the sequence. Loading of a file's part may take place at any time intervals. The separate parts do not have to be of identical size. The only limitation is their maximal size.

File loading starts with sending its first part to the server by using "UploadChunk" method with the parameter constituting of the "UploadChunkRequest" message. This message, in turn, shall contain an active ticket, XML string and "ChunkBag" element being a "bag" for subsequent parts of the file. While sending the first chunk of the file, the "ChunkBag" element shall remain empty, which will result in creation of a new "bag" and in its sending in the method's answer. "ChunkBag" is a structure containing information upon quantity of loaded parts, identifier of the "bag" and a total size of loaded data.

In order to put subsequent parts in the existing "bag" it is necessary to use the "UploadChunk" method once again, this time providing as the parameter the "UploadChunkRequest" message with a list of all data including the ones in the "ChunkBag". The service will return a modified "ChunkBag" structure in the result.

## 1.4. Processing of the tranche of orders

Processing of the tranche of orders takes place as the result of transferring the "CloseChunkBagRequest" parameter to the "CloseChunkBag" method of Siddin service. This parameter shall contain the following data: the ticket obtained during authentication, the "bag" with the loaded data, the version of

NICCI protocol, according to which the assembled file is being processed, and the optional own description of the tranche.

After this method is set off, the data loaded by the user are being consolidated and their correctness is being checked (in accordance with the XSD schema). In the event of error, a SOAP protocol exception (so-called SOAP FAULT) is being generated.

A consolidated tranche must be consistent with the NICCI protocol in version 2.1 or higher. The NICCI protocol has been described in the separate documents.

As a result of setting off the "CloseChunkBag", after tranche's correctness is checked, the user obtains the identifier of the ordered work (Job Id).

By using the "GetJobs" method, the user can debug the progress of works with the sent file. This method takes for the parameter the "GetJobsRequest" order, the parameters of which are: active ticket and, optionally, order's identifier. The result of setting off this method is a multi-element table of "Job" structures containing information upon progress, position in the queue of requested tranches (if parameter's value is 0 it means that the tranche is being currently processed), status (cancelled, an error occurred, finished, etc.), code of status, date of order, identifier, orderer's login and user description. If the identifier of the specific order has been given in the "GetJobsRequest" order, then the method shall return a single-element collection of "Job" elements with the data of the indicated order.

Processing of tranche's files may be cancelled by setting off the "CancelJob" method with "CancelJobRequest" parameter, which contains an active ticket and identifier of the performed work (Job Id). Order's cancellation can occur only for a tranche awaiting processing.

The SIDDIN server processes the tranche and prepares the answer in a form of XML file, containing the report from executed orders. The format of such report is consistent with the specification of the NICCI protocol in the version provided during ordering of the tranche.

The moment the status of performance downloaded by using "GetJobs" method will reach the value of "2" ("Processed"), it is possible to set off the "GetChunkBag" method with the "GetChunkBagRequest" parameter containing not only the active ticket but also the task's identifier and the size of a part of the answer. The last parameter determines the maximal size of the part of the answer obtained from a single download. In a particular case, this will only be one part.

After the method is set off, the answer will be divided and prepared to be downloaded by the user (the methods are described in the next subsection). This will result in returning of the chunk bag containing information upon quantity and size of the parts prepared for download.

In the event when the tranche has not been processed, and the "GetChunkBag" method is set off, an error will be generated.

In case of critical errors (e.g. non-recognized version of NICCI protocol), the NICCI server returns a SOAP protocol exception (so-called SOAP FAULT), and the ordered file is being provided with the "FAILED" status.

## 1.5. Download of the answers from the service

In order to download a result of tranche processing from the service, it is necessary to use the chunk bag (“ChunkBag”) obtained after setting off the “GetChunkBag” and by “DownloadChunk” method. This method downloads the “DownloadChunkRequest” message with three parameters: the active ticket, the chunk bag, a part of which is to be downloaded, and the number of the part which shall be downloaded. The result of setting off the method is a string containing a part – or in a particular case – the entire answer. It is a report from executed orders (format of such report is described in the NICCI protocol).

The downloaded parts are not being removed from the server immediately, so it is possible to get them repeatedly if such necessity occurs.

## 1.6. Data security

Before processing, an XML document kept in the KRD system and previously sent by the user, is being tested for its compatibility with the NICCI protocol schema and then digitally signed (together with the name of the user, who put it in the system), to assure that the document will not get modified during waiting for the processing.

## 1.7. Error service

All errors reported by the SIDDIN service are being remembered on the server, and the user only obtains a general error “SOAP FAULT”. However, in the description of this error, there is the error’s identifier that will allow the IT department of the National Debt Register (KRD) to localize the error’s description on server’s site. That is why in case of any problems it is recommended to contact a KRD adviser (e.g. pomocit@krd.pl) providing identifier of the error.

# 2. WSDL file

WSDL file (Web Services Description Language) describes types of operations which can be performed on web service by using SOAP. The file contains descriptions of data types (types), types of messages accepted by the service (messages), names and parameters of operations (operations) and attributes of the service. The WSDL file for the SIDDIN service is available under the following addresses:

- https://demo.krd.pl/Siddin/2.1/Import.asmx?WSDL – demo version (application for a free demo account for testing can be sent **here**)
- https://services.krd.pl/Siddin/2.1/Import.asmx?WSDL – production version

## 2.1. *Definitions of types*

### 2.1.1. Type: SignedRequest

This type is a basis for most types defining parameters of service's methods, except from the "Login" method. It contains information upon active ticket of the user.

```
<s:complexType name="SignedRequest">
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="Ticket" type="s:string"/>
 </s:sequence>
</s:complexType>
```

### 2.1.2. Type: LoginRequest

The "LoginRequest" type is the parameter of the "Login" method of the service and is used for transferring information upon user's name and password.

```
<s:complexType name="LoginRequest">
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="UserName" type="s:string"/>
  <s:element minOccurs="0" maxOccurs="1" name="Password" type="s:string"/>
 </s:sequence>
</s:complexType>
```

### 2.1.3. Type: LogoutRequest

This type is used for logging out of the service, and it "packs" only the *SignedRequest* type.

```
<s:complexType name="LogoutRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest"/>
 </s:complexContent>
</s:complexType>
```

### 2.1.4. Type: ChunkBag

The "ChunkBag" type describes parameters of the "bag" into smaller parts of the large file of the tranche. It contains such elements as "ID", that is object's identifier, "Count" determining quantity of loaded parts, "Size" transferring the information upon total size of loaded parts in bites, and "NicciVersion" with the information upon protocol's version. This type is used in "UploadChunkBag", "DownloadChunk" methods and as a return parameter of "GetChunkBag" method.

```
<s:complexType name="ChunkBag">
 <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="ID" type="s2:guid"/>
  <s:element minOccurs="1" maxOccurs="1" name="Count" type="s:int"/>
  <s:element minOccurs="1" maxOccurs="1" name="Size" type="s:long"/>
  <s:element minOccurs="1" maxOccurs="1" name="NicciVersion" type="s1:NicciVersionEnum"/>
 </s:sequence>
</s:complexType>
```

### 2.1.5. Type: NicciVersionEnum

Enumerated type, determining versions of the NICCI protocol that can be transferred through the SIDDIN service. Version 2.1 of the service supports all available specifications of NICCI – 1.1 (protocol withdrawn from use), 1.2 (protocol withdrawn from use), 2.0 (protocol withdrawn from use), 2.1, 3.0, 3.1, and 3.2.

```
<s:simpleType name="NicciVersionEnum">
 <s:restriction base="s:string">
  <s:enumeration value="NotSpecified"/>
  <s:enumeration value="Version_1_1"/>
  <s:enumeration value="Version_1_2"/>
  <s:enumeration value="Version_2_0"/>
  <s:enumeration value="Version_2_1"/>
  <s:enumeration value="Version_3_0"/>
  <s:enumeration value="Version_3_1"/>
  <s:enumeration value="Version_3_2"/>
 </s:restriction>
</s:simpleType>
```

### 2.1.6. Type: Job

Variable of "Job" type occurs as the element of the collection being returned as a result of setting off the "GetJobs" method. It contains such information as: order's number (JobID), another order's number (JobNumber), number in the queue for processing (QueueNumber), percentage of progress (Progress), login of the orderer (QueueLogin), order's date (QueueDate), code of order's status (StatusCode) consistent with the *JobStatusCodeEnum* type, status in verbal form (Status), and own (optional) description of the tranche / executed order (Description).

```
<s:complexType name="Job">
 <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="JobNumber" type="s:int"/>
  <s:element minOccurs="1" maxOccurs="1" name="QueueNumber" type="s:int"/>
  <s:element minOccurs="1" maxOccurs="1" name="QueueDate" type="s:dateTime"/>
  <s:element minOccurs="0" maxOccurs="1" name="Status" type="s:string"/>
  <s:element minOccurs="1" maxOccurs="1" name="StatusCode" type="s1:JobStatusCodeEnum"/>
  <s:element minOccurs="1" maxOccurs="1" name="Progress" type="s:int"/>
  <s:element minOccurs="0" maxOccurs="1" name="QueueLogin" type="s:string"/>
  <s:element minOccurs="1" maxOccurs="1" name="JobID" type="s2:guid"/>
  <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string"/>
 </s:sequence>
</s:complexType>
```

### 2.1.7. Type: JobStatusEnum

This type defines all possible states, in which the tranche ordered by the user can be found. The order's status is a part of the "Job" structure being returned after the "GetJobs" method is set off.

“Queued” state means that the tranche is awaiting processing (the “QueueNumber” element being returned with the status by “QueryJob” determines the tranche’s number in the queue).

“Cancelled” state means that the user removed the tranche from the queue by using “Cancel” method of the server.

“Started” state means that the tranche is being currently processed.

“Processed” state means that processing of the tranche has been finished (in entirety or partially, after setting off the “Cancel” method of the server).

“Failed” state means that the tranche could not have been processed for the reasons independent of the user. This state occurs only in the event of critical errors. Errors in processing of separated tranche orders are being returned to the report in accordance with XSD schema of a chosen version of NICCI.

```
<s:simpleType name="JobStatusCodeEnum">
 <s:restriction base="s:string">
  <s:enumeration value="CANCELED"/>
  <s:enumeration value="FAILED"/>
  <s:enumeration value="PROCESSED"/>
  <s:enumeration value="QUEUED"/>
  <s:enumeration value="STARTED"/>
 </s:restriction>
</s:simpleType>
```

### 2.1.8. Type: UploadChunkRequest

*UploadChunkRequest* is a type used for sending a part of a file to the server. It consists of the “Data” parameter, containing a string of a sent part of a file, and “ChunkBag” parameter described in the subsection **2.1.4**. While sending the first part of a file, we leave the “ChunkBag” empty, which leads to generation of a new "bag" for files. This new "bag" is what we obtain as the result of processing of the service’s method. Since this type extends the *SignedRequest* type, it also requires providing the active ticket.

```
<s:complexType name="UpladChunkRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ChunkBag" type="s1:ChunkBag"/>
    <s:element minOccurs="0" maxOccurs="1" name="Data" type="s:string"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.9. Type: CloseChunkRequest

We use this type as a parameter of the “CloseChunkBag” method, ending the process of sending data to the service. Apart from the active ticket, the *CloseChunkRequest* type has also got such parameters as “ChunkBag”, “Description” containing an optional description of the executed order (the description of the order shall consist of up to 512 keys and will appear on the order list in the “Job” element), and “GetDifference” (the so-called differential tranche, coordinates with Nicci 2.1).

```
<s:complexType name="CloseChunkBagRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ChunkBag" type="s1:ChunkBag"/>
    <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string"/>
    <s:element minOccurs="1" maxOccurs="1" name="GetDifference" type="s:boolean"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.10. Type: GetChunkBagRequest

This type is used for dividing an output file into parts of a defined size and to download data of the "ChunkBag", into which the parts of the divided file have been put. Apart from the "Ticket" parameter resulting from the extension of *SignedRequest* type, the *GetChunkBagRequest* type has also got the "JobId" parameter with order's identifier and "ChunkSize" determining a size of a single chunk of the downloaded file.

```
<s:complexType name="GetChunkBagRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="JobId" type="s2:guid"/>
    <s:element minOccurs="1" maxOccurs="1" name="ChunkSize" type="s:int"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.11. Type: DownloadChunkRequest

This type is used in the "DownloadChunk" method and serves for transferring information upon the "bag" for files ("ChunkBag") and the number of file's part to be downloaded ("Number"). Of course, it also requires providing the active ticket of the Client, for it is an extension of the *SignedRequest* type.

```
<s:complexType name="DownloadChunkRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ChunkBag" type="s1:ChunkBag"/>
    <s:element minOccurs="1" maxOccurs="1" name="Number" type="s:int"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.12. Type: GetJobsRequest

The *GetJobsRequest* type is used for downloading the list of orders kept on the server. Its only additional parameter (apart from the "Ticket") is the "JobId" with identifier of the order whose status is to be downloaded. The "JobId" element is required, it can however remain empty. In such case, the method will return the list of all Client's orders kept on the server.

The "GetAll" element works similarly – it downloads the entire queue of ordered XML files (within a single Client account). To download the entire queue of ordered XML files, specify *true* in the "GetAll" element. As a result of processing, we will get a list of *Job* elements. You can narrow down this result using the "LastGet" element, then after processing we will get a list of *Job* elements whose status has changed since the date specified in the "LastGet" element. Alternatively, you can use the "LastNumber" element, where we specify the job number ("JobNumber"). As a result of processing, we will get a list of *Job* elements whose status has changed since the specified order was submitted.

```
<s:complexType name="GetJobsRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="GetAll" type="s:boolean"/>
    <s:element minOccurs="1" maxOccurs="1" name="LastGet" type="s:dateTime"/>
    <s:element minOccurs="0" maxOccurs="1" name="LastNumber" type="s:int"/>
    <s:element minOccurs="1" maxOccurs="1" name="JobId" type="s2:guid"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.13. Type: ArrayOfJob

The *ArrayOfJob* is nothing other than a table of *Job* types containing information upon orders. If we provide order's identifier in the *GetJobsRequest* type, the *ArrayOfJob* will only contain one *Job* element.

```
<s:complexType name="ArrayOfJob">
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="unbounded" name="Job" nillable="true" type="s1:Job"/>
 </s:sequence>
</s:complexType>
```

### 2.1.14. Type: CancelJobRequest

This type is used for cancelling the order which has not been processed yet. A parameter of this type is the identifier of the order "JobId" and user's active ticket "Ticket" resulting from inheriting from the *SignedRequest*.

```
<s:complexType name="CancelJobRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="JobId" type="s2:guid"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.15. Type: RemoveChunkBagRequest

This type is used to delete the uploaded "bag" ("ChunkBag"). The parameter of this type is the identifier of the "bag" ("ChunkBagId") and the active user ticket ("Ticket").

```
<s:complexType name="RemoveChunkBagRequest">
 <s:complexContent mixed="false">
  <s:extension base="tns:SignedRequest">
   <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="ChunkBagId" type="s2:guid"/>
   </s:sequence>
  </s:extension>
 </s:complexContent>
</s:complexType>
```

### 2.1.16. Type: guid

Type of unique strings used for unequivocal identification of tasks and "bags". Elements of this type are generated by the server (pack's number or order's number) and the user does not have to care about their uniqueness.

```
<s:simpleType name="guid">
 <s:restriction base="s:string">
  <s:pattern value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-
F]{12}"/>
</s:restriction>
</s:simpleType>
```

## 2.2. Definitions of messages

WSDL file for the SIDDIN service defines 26 messages: input and output message for each method of the service. These are the following messages:

- `LoginSoapIn` and `LoginSoapOut` for the `Login` method
- `LoginExSoapIn` and `LoginExSoapOut` for the `LoginEx` method
- `LogoutSoapIn` and `LogoutSoapOut` for the `Logout` method
- `UploadChunkSoapIn` and `UploadChunkSoapOut` for the `UploadChunk` method
- `CloseChunkBagSoapIn` and `CloseChunkBagSoapOut` for the `CloseChunkBag` method
- `CloseChunkBagExSoapIn` and `CloseChunkBagExSoapOut` for the `CloseChunkBagEx` method
- `GetChunkBagSoapIn` and `GetChunkBagSoapOut` for the `GetChunkBag` method
- `GetChunkBagExSoapIn` and `GetChunkBagExSoapOut` for the `GetChunkBagEx` method
- `GetRepeatFileSoapIn` and `GetRepeatFileSoapOut` for the `GetRepeatFile` method
- `DownloadChunkSoapIn` and `DownloadChunkSoapOut` for the `DownloadChunk` method
- `GetJobsSoapIn` and `GetJobsSoapOut` for the `GetJobs` method
- `CancelJobSoapIn` and `CancelJobSoapOut` for the `CancelJob` method
- `RemoveChunkBagSoapIn` and `RemoveChunkBagSoapOut` for the `RemoveChunkBag` method

The parameters of setting off and the types of returned values (defined e.g. in the previous section) are being defined for each of these messages. Below is a description of the basic methods (the `LoginEx`, `CloseChunkBagEx`, `GetChunkBagEx` methods were created at one time for one of our applications – Sabar and the XML Nicci 2.1 / CSV Yonick 2.0 protocol. The same is true of the `GetRepeatFile` method, which downloads the so-called replay tranche only with incorrect items that the system rejected, processing the batch tranche – only that it also works with other versions of the Nicci protocol).

### 2.2.1. Messages: LoginSoapIn and LoginSoapOut

The input message for the "Login" method of *LoginRequest* type contains information upon identifier ("UserName") and password ("Password") of the user.

The output message returns the *LoginResult* in a form of string – ticket.

```
<wsdl:message name="LoginSoapIn">
 <wsdl:part name="LoginRequest" element="tns:LoginRequest"/>
</wsdl:message>
<wsdl:message name="LoginSoapOut">
 <wsdl:part name="LoginResult" element="tns:LoginResult"/>
</wsdl:message>
```

### 2.2.2. Messages: LogoutSoapIn and LogoutSoapOut

The input message for the "Logout" method requires providing the *LogoutRequest* type with an active ticket. This method does not return any parameters.

```
<wsdl:message name="LogoutSoapIn">
 <wsdl:part name="LogoutRequest" element="tns:LogoutRequest"/>
</wsdl:message>
<wsdl:message name="LogoutSoapOut"/>
```

### 2.2.3. Messages: UploadChunkSoapIn and UploadChunkSoapOut

The *UploadChunkSoapIn* is a message of the "UploadChunk" method used for transferring the part of XML file of the ordered NICCI tranche. Requires providing the *UploadChunkRequest* described in section **2.1.8**. As the answer, we get the *UploadChunkResult* element, containing a "bag" to put the parts of the tranche in.

```
<wsdl:message name="UploadChunkSoapIn">
 <wsdl:part name="UpladChunkRequest" element="tns:UpladChunkBagRequest"/>
</wsdl:message>
<wsdl:message name="UploadChunkSoapOut">
 <wsdl:part name="UploadChunkResult" element="s1:UploadChunkResult"/>
</wsdl:message>
```

### 2.2.4. Messages: CloseChunkBagSoapIn and CloseChunkBagSoapOut

In order to finish sending of the file's part and to assign its execution, we have to set off the "CloseChunkBag" method, which takes for parameter the element of the *CloseChunkBagRequest* type.

The result of setting off this method is an element of the *CloseChunkBagRequest* type containing order's identifier "JobId".

```
<wsdl:message name="CloseChunkBagSoapIn">
 <wsdl:part name="closeChunkBagRequest" element="tns:CloseChunkRequest"/>
</wsdl:message>
<wsdl:message name="CloseChunkBagSoapOut">
 <wsdl:part name="CloseChunkBagResult" element="tns:CloseChunkBagResult"/>
</wsdl:message>
```

### 2.2.5. Messages: GetChunkBagSoapIn and GetChunkBagSoapOut

In order to download the answer file after finishing tranche's processing, we have to set off the "GetChunkBag" method first. The parameter of setting off is the *GetChunkBagRequest*. The result of this operation is an element of the *GetChunkBagResult* type including the “ChunkBag” necessary to download the answer by using the “DownloadChunk” method.

```
<wsdl:message name="GetChunkBagSoapIn">
 <wsdl:part name="GetChunkBagRequest" element="tns:GetChunkBagRequest"/>
</wsdl:message>
<wsdl:message name="GetChunkBagSoapOut">
 <wsdl:part name="GetChunkBagResult" element="s1:GetChunkBagResult"/>
</wsdl:message>
```

### 2.2.6. Messages: DownloadChunkSoapIn and DownloadChunkSoapOut

In order to obtain the results of processing, the user shall set off the “DownloadChunk” method. The parameter of setting off this method is the “DownloadChunkRequest” of the *DownloadChunkRequest* type described in the section **2.1.11**. The result of setting off the method is the *DownloadChunkResult* in a form of string containing a part (or particularly whole) XML file.

```
<wsdl:message name="DownloadChunkSoapIn">
 <wsdl:part name="downloadChunkRequest" element="tns:DownloadChunkRequest"/>
</wsdl:message>
<wsdl:message name="DownloadChunkSoapOut">
 <wsdl:part name="DownloadChunkResult" element="tns:DownloadChunkResult"/>
</wsdl:message>
```

### 2.2.7. Messages: GetJobsSoapIn and GetJobsSoapOut

The user has got a possibility to check the status of the assigned task by setting off the “GetJobs” method. Its setting off parameter is an element of the *GetJobsRequest* consisting of: an active ticket (“Ticket”) and identifier of the element in the queue (“JobId”).

The resulting parameter is an element of the *GetJobsResult* type, containing the table of *Job* type elements.

```
<wsdl:message name="GetJobsSoapIn">
 <wsdl:part name="GetJobsRequest" element="tns:GetJobsRequest"/>
</wsdl:message>
<wsdl:message name="GetJobsSoapOut">
 <wsdl:part name="GetJobsResult" element="tns:GetJobsResult"/>
</wsdl:message>
```

### 2.2.8. Messages: CancelJobSoapIn and CancelJobSoapOut

The *CancelJobSoapIn* message serves as a parameter of the “CancelJob” method excluding the ordered task from the queue, and it contains the *CancelJobRequest* type, an active ticket (“Ticket”) and identifier of the element in the queue (“JobId”).

The method does not return any data, therefore it is necessary to use the “GetJobs” method, providing order’s identifier, to check whether the order has been cancelled or not.

```
<wsdl:message name="CancelJobSoapIn">
 <wsdl:part name="CancelJobRequest" element="tns:CancelJobRequest"/>
</wsdl:message>
<wsdl:message name="CancelJobSoapOut"/>
```

#### 2.2.9. Messages: RemoveChunkBagSoapIn and RemoveChunkBagSoapOut

The *RemoveChunkBagSoapIn* message serves as a parameter to the “RemoveChunkBag” method that removes the uploaded "bag" (“ChunkBag”).

In order to remove the uploaded "bag", set off the “RemoveChunkBag” method. For the operation to succeed, you must specify the active ticket (“Ticket”) and the Id of the "bag" we want to remove. The method does not return any parameters. Informatively – "bags" are periodically removed on our side anyway.

```
<wsdl:message name="RemoveChunkBagSoapIn">
 <wsdl:part name="removeChunkBagRequest" element="tns:RemoveChunkBagRequest"/>
</wsdl:message>
<wsdl:message name="RemoveChunkBagSoapOut"/>
```

### 2.3. *portType Element*

The next element of WSDL file (portType) defines a set of operations available within the service – each of the components contains, related to it, input and output messages (the definitions of these messages can be found in the previous section) and a name.

```
 <wsdl:portType name="Import">
  <wsdl:operation name="Login">
   <wsdl:input message="tns:LoginSoapIn"/>
   <wsdl:output message="tns:LoginSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="LoginEx">
   <wsdl:input message="tns:LoginExSoapIn"/>
   <wsdl:output message="tns:LoginExSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="Logout">
   <wsdl:input message="tns:LogoutSoapIn"/>
   <wsdl:output message="tns:LogoutSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="UploadChunk">
   <wsdl:input message="tns:UploadChunkSoapIn"/>
   <wsdl:output message="tns:UploadChunkSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="CloseChunkBag">
   <wsdl:input message="tns:CloseChunkBagSoapIn"/>
   <wsdl:output message="tns:CloseChunkBagSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="CloseChunkBagEx">
   <wsdl:input message="tns:CloseChunkBagExSoapIn"/>
   <wsdl:output message="tns:CloseChunkBagExSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetChunkBag">
   <wsdl:input message="tns:GetChunkBagSoapIn"/>
   <wsdl:output message="tns:GetChunkBagSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetChunkBagEx">
   <wsdl:input message="tns:GetChunkBagExSoapIn"/>
   <wsdl:output message="tns:GetChunkBagExSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetRepeatFile">
   <wsdl:input message="tns:GetRepeatFileSoapIn"/>
   <wsdl:output message="tns:GetRepeatFileSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="DownloadChunk">
   <wsdl:input message="tns:DownloadChunkSoapIn"/>
   <wsdl:output message="tns:DownloadChunkSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetJobs">
   <wsdl:input message="tns:GetJobsSoapIn"/>
   <wsdl:output message="tns:GetJobsSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="CancelJob">
   <wsdl:input message="tns:CancelJobSoapIn"/>
   <wsdl:output message="tns:CancelJobSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="RemoveChunkBag">
   <wsdl:input message="tns:RemoveChunkBagSoapIn"/>
   <wsdl:output message="tns:RemoveChunkBagSoapOut"/>
  </wsdl:operation>
 </wsdl:portType>
```

## 2.4. Binding element

The next element of the WSDL file (binding) defines connection between services available within the service and the transport protocol. By using the `soap:binding` element, the transport’s schema and style are being chosen. Then the subsequent parameters (e.g. address and action’s style) are being determined for each method of the service (operation).

```
<wsdl:binding name="Import" type="tns:Import">
 <wsdl:documentation>
  <wsi:Claim conformsTo="http://ws-i.org/profiles/basic/1.1"/>
 </wsdl:documentation>
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="Login">
  <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/Login"
style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="LoginEx">
  <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/LoginEx"
style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="Logout">
  <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/Logout"
style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="UploadChunk">
  <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/UploadChunk"
style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
```

```
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="CloseChunkBag">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/CloseChunkBag"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="CloseChunkBagEx">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/CloseChunkBagEx"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetChunkBag">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/GetChunkBag"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetChunkBagEx">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/GetChunkBagEx"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetRepeatFile">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/GetRepeatFile"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="DownloadChunk">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/DownloadChunk"
style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetJobs">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/GetJobs"
 style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="CancelJob">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/CancelJob"
 style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="RemoveChunkBag">
   <soap:operation soapAction="http://Siddin.ServiceContracts/2006/09/RemoveChunkBag"
 style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
```

## 2.5. Service definition

The service definition includes determination of a port – a location, where the service is accessible and choosing one of the previously defined connections for given port.

```
 <wsdl:service name="Import">
  <wsdl:port name="Import" binding="tns:Import">
   <soap:address location="https://services.krd.pl/Siddin/2.1/Import.asmx"/>
  </wsdl:port>
 </wsdl:service>
```

**END OF DOCUMENT**