mission‹one› GmbH

Reference Document		
		Page 1 of 44

mission‹one›control - Customer Interface API V2

Author: mission‹one› GmbH	Revision No: 7
Date: 18.08.2016	Attachments: Program Examples

Content

1. Introduction

mission<one>control (hereinafter referred to as mission<cl> or cl) is the in-house solution of the mission<one> GmbH for professional email marketing and is specially made for the realization of extensive marketing campaigns.

As part of this system mission<one> offers its customers the module 'API'. It enables you to get access to certain functions of the mission<cl> and to integrate it into own products.

What does the API offer?

A lot! Mainly it is used for the following purposes:

- Automatic synchronization of subscriber lists from the mission<cl> with your own database or your own CRM-systems.
- Creation of person groups directly out of your own software solution.
- Export of subscriber data from the mission<cl> and subsequently the transfer to your own system.
- The dispatch of newsletters to single subscribers or whole person groups (function is not available on first version)
- Import and export of newsletter contents (function is not available on first version).

How does API work?

The mission<cl> Customer Interface API is a web service based on the SOAP report and enables you to easily integrate the API method and its functions into your own projects. The web service is accessible via internet. An access protection is regulated by an authentication function (see 3.2).

The API is available via the following URL:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx

1.1 Glossary

- mission<cl> → mission<one>control is the latest version of the mission<one> email marketing software

- API → An application programming interface (API) is an interface implemented by a software program which enables it to interact with other software. It facilitates interaction between different software programs similar to the way the user interface facilitates interaction between humans and computers.

- Web services → are typically application programming interfaces (API) or web APIs that are accessed via Hypertext Transfer Protocol and executed on a remote system hosting the requested services. Web services tend to fall into one of two camps: Big Web Services and RESTful Web Services. "Big Web Services" use Extensible Markup Language (XML) messages that follow the Simple Object Access Protocol (SOAP) standard and have been popular with traditional enterprise.

- SOAP → originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks.

- Client: is an application or system that accesses a remote service on another computer system, known as a server, by way of a network.

- Subscriber → recipient of the newsletter

- Attribute → Attributes define the structure, fields or columns of the subscriber database. They are freely definable in the application. We describe attributes within API V2 as an array of Fields objects.

- MID → Mid is unique key you get from personalized link within newsletter. It contains information's about subscriber which received newsletter.

2. Methods of mission<one>control-API V2

This part describes the function perimeters of the API V2. The functionality is divided in the following parts:

- Attributes methods
- Blacklists methods
- Person Groups methods
- Subscribers methods

- Statistics methods
- Subscribers import methods
- Others

Version 1.

2.1 Attributes methods

These methods are mainly used for retrieving the attributes and possible attributes values and corresponding data types.

2.1.1 Method: *AttributesGet*

Usage: Returns attribute details for one or more attributes, depending on incoming parameter provided.
Address: https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=AttributesGet
Input Parameters: a list of attributesId's like gender, name, lastname, etc. Or nothing if you want all attributes returned.
Response: An array of detailed attributes objects like this:

```
<CustomerAttribute>
  <Name>string</Name>
  <DataType>string</DataType>
  <DisplayName>string</DisplayName>
  <Description>string</Description>
  <DefaultValues>string</DefaultValues>
  <StringLength>int</StringLength>
  <IsRange>boolean</IsRange>
  <RangeFrom>string</RangeFrom>
  <RangeTo>string</RangeTo>
  <HasRegularExpressions>boolean</HasRegularExpressions>
  <RegularExpression>string</RegularExpression>
  <IsUnique>boolean</IsUnique>
  <IsMandatory>boolean</IsMandatory>
  <HasAllowedValues>boolean</HasAllowedValues>
  <AllowedValues>
    <AttributeValues xsi:nil="true" />
    <AttributeValues xsi:nil="true" />
  </AllowedValues>
 </CustomerAttribute>
```

Common error: none;

2.1.2 Method: *AttributesCustomGet*

Usage: Returns custom attribute list.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=AttributesCustomGet

Input Parameters: None;
Response: An array of detailed attributes objects like this:

```
  <AttributesCustomGetResult>
    <string>string</string>
    <string>string</string>
  </AttributesCustomGetResult>
```

Common error: none;

2.1.3 Method: *AttributeSingleSelectValuesGet*

Usage: Returns any singleselect attribute type possible values list.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=AttributeSingleSelectValuesGet
Input Parameters: Attribute name (e.g. gender, colors etc.); - Mandatory
Response: An array of Attribute value objects like this:

```
      <AttributeValues>
          <AttributeId>int</AttributeId>
         <AttributeValue>string</AttributeValue>
         <Culture>string</Culture>
    </AttributeValues>
```

Common error: none;

2.1.4 Method: *AttributeMultiSelectValuesGet*

Usage: Returns any multiselect attribute type possible values list.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=AttributeMultiSelectValueGet
Input Parameters: Attribute name (e.g. gender, colors, version etc.); - Mandatory
Response: An array of Attribute value objects like this:

```
      <AttributeValues>
          <AttributeId>int</AttributeId>
         <AttributeValue>string</AttributeValue>
         <Culture>string</Culture>
    </AttributeValues>
```

Common error: none;

2.2 Subscriber methods

These methods are used for the administration and maintenance of the subscriber data. (newsletter recipients). One can use this method to do the following operations: Delete Subscribers, Edit Subscribers, Insert Subscribers, Export Subscribers and methods for usage by MID.

2.2.1 Method *SubscriberActivateMid*

Usage: Activates subscriber from MID provided.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscriberMidActivate
Input Parameters: Mid – string (see glossary) - Mandatory
Response: An array of Attribute value objects like this:

```
<SubscriberMidActivateResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
   </SubscriberMidActivateResult>
```

Common error: If you provide invalid MID – you will usually get the “Link not personalized” error.

2.2.2 Method *SubscriberMidDelete*

Usage: Deletes subscriber from MID provided.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscriberMidDelete
Input Parameters: Mid – string (see glossary) - Mandatory
Response: An array of Attribute value objects like this:

```
<SubscriberMidActivateResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
   </SubscriberMidActivateResult>
```

Common error: If you provide invalid MID – you will usually get the “Link not personalized” error. Additionally if the subscriber is already deleted you might get “Subscriber does not exist error”.

2.2.3 Method *SubscriberMidGet*

Usage: Returns subscriber with all details from MID provided.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscriberMidGet
Input Parameters: Mid – string (see glossary) - Mandatory
Response: An array of Attribute value objects like this:

```
<SubscriberMidActivateResult>
      <IsError>boolean</IsError>
      <ErrorMessage>string</ErrorMessage>
      <SucceededSubscribersCount>int</SucceededSubscribersCount>
      <FailedSubscribers>
        <FailedSubscriber>
          <SourceSubscriber xsi:nil="true" />
          <ErrorMessage>string</ErrorMessage>
        </FailedSubscriber>
        <FailedSubscriber>
          <SourceSubscriber xsi:nil="true" />
          <ErrorMessage>string</ErrorMessage>
        </FailedSubscriber>
      </FailedSubscribers>
    </SubscriberMidActivateResult>
```

Common error: If you provide invalid MID – you will usually get the “Link not personalized” error. Additionally if the subscriber is already deleted you might get “Subscriber does not exist error”.

2.2.4 Method *SubscribersInsert*

Usage: Inserts Subscribers to the system. Optionally, inserts subscriber to existing or newly provided PersonGroups and sends newsletters to subscribers.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersInsert
Input Parameters:

- Subscribers[] (mandatory) - An Array of subscriber objects. Subscriber Object is complex data type it will be described additionally. Subscriber object is containing following attributes:
    - Head – String type - referenced by your system so you can track insertion process. E.g. you can put your system serial number inside head, and if subscriber fails this data will persist within failed-subscribers list.
    - Where– complex type array. Interest type definition is contains array of Condition objects:

```
<s:complexType name="Condition">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="Operator" type="tns:ConditionOperator"/>
    <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string"/>
  </s:sequence>
</s:complexType>
```

Condition is used widely on CI API V2. It's always representing unique recognition for single subscriber. E.g. you can send their email and project within condition array with update command and it will update exactly that subscriber.
This parameter in SubscriberInsert method is used when you want to add existing subscriber to additional projects (Multiproject usage).

- Interests []– complex type array. Interest type definition is:

```
<s:complexType name="Interest">
   <s:sequence>
     <s:element minOccurs="1" maxOccurs="1" name="InterestAction" type="tns:Action" />
      <s:element minOccurs="1" maxOccurs="1" name="Value" type="s:int" />
   </s:sequence>
</s:complexType>
```

InterestAction is yet another complex list, one can choose between Insert or Update values for this object.

So the common usage explanation and relation between this and parent object is:
SUBSCRIBER->INTERESTS->INTEREST[]

- Fields[] – complex type Array. Field type is representing attribute name and value and its definition is:

```
<s:complexType name="Field">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
  </s:sequence>
</s:complexType>
```

So the common usage explanation and relation between this and parent object is:
SUBSCRIBER->FIELDS->FIELD[]

- MultiSelectFields - is representing another complex type array. Multi Select field for subscriber means it can contain more of the values for one special attribute (e.g. Color; subscriber can choose red, blue and orange at once. Unline single select types where we can only choose one value.
Its definition is:

```
<s:complexType name="MultiSelectField">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="Values" type="tns:ArrayOfFieldValue" />
      <s:element minOccurs="1" maxOccurs="1" name="InsertValueIfNotExists" type="s:boolean" />
    </s:sequence>
 </s:complexType>
```

So the common usage explanation and relation between this and parent object is:
SUBSCRIBER->MULTISELECTFIELDS->MULTISELECTFIELD[]

NOTE: using this field you can also insert non-existing values. E.g. you can send value for color "Black" and it will be inserted into system for further use.

- InsertType – The type of insert. Possible values are:
  - Normal – Inserts subscriber with given attributes, if subscriber already exists return error
  - InsertOrOverwrite – Inserts subscriber with given attributes, if subscriber already exists method overwrites existing subscriber. The subscribers history remains, but all data which is not send is deleted (Subscriber will have only new data). PLEASE NOTE: If you want to update existing data, then use SubscribersInsertOrUpdate method instead!
  - InsertWithDeleteIfFound – Inserts subscriber with given attributes, if subscriber already exists method first deletes existing subscriber and then inserts given subscriber
- PersongGroups – An array of PersonGroups2Subscriber object. It is complex type object, it will be described additionally.
- AfterInsertAction – You can choose action after insert is done. It can be: None or SigninNewsletter or DoiNewsletter or ManualNewsletter
- ManualNewsletterId – Newsletter id that will be sent after insertion. It's used only if previous parameter is set to "ManualNewsletter".
  You can combine "AfterInsertAction"="DoiNewsletter and "ManualNewsletterId" when you want to send some special newsletter as a Doi notification. If ManualNewsletterId is not set, then the software will send DoiNewsletter which is specified in project.

Minimum insert:
- To insert subscriber, you minimally need to send fields prj_projectid and email, with all mandatory fields you set on your system. You also need to set-up interests, and if no interests are needed, you need to set at least 0 (zero) value interests.

Multi-project usage: This method can be used for multi-project subscriber feature. If you want to insert subscriber in many projects, simply send many all projects that are your intention within Fields array and subscriber will be applied to all of them.
  - Subscriber->Fields->Field-> {prj_projectid, 3}; Subscriber->Fields->Field-> {prj_projectid, 4} Subscriber->Fields->Field-> {prj_projectid, 6}

If you want to add subscriber to another project (s) later, you need to send Subscriber->Where condition containing publicid, and within fields again you need to send projects you want subscriber to be applied.

Response: An array of Attribute value objects like this:
```
<SubscribersInsertResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
```

```
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
  </SubscribersInsertResult>
```

Common error: If one or many of input subscriber's objects are invalid for some reason (e.g. invalid field sent, double-email or blacklisted email, or missing email) you will find your subscriber within *FailedSubscribers* array and inside each of subscriber you can find property: Subscriber->ErrorMessage explaining the reason why the insert action failed.

2.2.5 Method *SubscribersUpdate*

Usage: Updates existing subscribers elements within system (e.g. attribute values, interests, persongroups etc.)
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersUpdate
Input Parameters:

- Subscribers[] (mandatory) - An Array of subscriber objects. Subscriber Object is complex data type it will be described additionally. Subscriber object is described detailed within SubscriberInsert method description
- PersongGroups – An array of PersonGroups2Subscriber object. It is complex type object, it will be described additionally.
- AfterUpdateAction – You can choose action after update is done. It can be: None or SendNewsletter;
- ManualNewsletterId – Newsletter id that will be sent after insertion. It's used only if previous parameter is set to "SendNewsletter"

Multi-project usage: No "special" usage of multi-projects feature is required. If you update subscriber on one project, it will be applied to all projects.
Response: An array of Attribute value objects like this:

```
<SubscribersInsertResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
  </SubscribersInsertResult>
```

Common error: If one or many of input subscriber's objects are invalid for some reason (e.g. invalid field sent, double-email or blacklisted email) you will find your subscriber within FailedSubscribers array and inside each of subscriber you can find property: Subscriber->ErrorMessage explaining the reason why the insert action failed. Also watch out that you send subscriber->Where that have uniqueness within condition array, otherwise it will respond with an error "Subscriber does not exist".

2.2.6 Method *SubscribersDelete*

Usage: Deletes existing subscribers from system. You only need to send correct Subscriber->Where describing subscriber as unique element. Eg some unique field or combination of project_id and email.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersDelete
Input Parameters:

- Subscribers[] (mandatory) - An Array of subscriber objects. Subscriber Object is complex data type it will be described additionally. Subscriber object is described detailed within SubscriberInsert method description
- DeleteType – Choose between Deactivate, Delete and DeleteInactiveDOI (this delete type is used to delete inactive subscribers that are still in DOI process)
- afterDeleteAction– You can choose action after delete is done. It can be: None or SendSignoutNewsletter or SendManuallNewsletter;
- ManualNewsletterId – Newsletter id that will be sent after insertion. It's used only if previous parameter is set to "SendManuallNewsletter"

Response: An array of Attribute value objects like this:

```
<SubscribersInsertResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
  </SubscribersInsertResult>
```

Common error: Watch out that you send subscriber->Where that have uniqueness within condition array, otherwise it will respond with an error "Subscriber does not exist".

2.2.7 Method *SubscribersInsertOrUpdate*

Usage: InsertOrUpdate method chooses whether send subscribers are for update or insert action for you. Its usefull for synchronization operations. Usage is analog to the calls for SubscriberUpdate or SubscribersInsert as well as the input parameters
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersInsertOrUpdate

Input Parameters:

- Subscribers[] (mandatory) - An Array of subscriber objects. Subscriber Object is complex data type it will be described additionally. Subscriber object is described detailed within SubscriberInsert method description
- InsertType– as in insert method.
- AfterUpdateAction– as with update method
- AfterInsertAction - similar to insert method
- ManualNewsletterId – Used only if you specified it, as with insert or update methods.

Response: An array of Attribute value objects like this:

```
<SubscribersInsertResult>
    <IsError>boolean</IsError>
    <ErrorMessage>string</ErrorMessage>
    <SucceededSubscribersCount>int</SucceededSubscribersCount>
    <FailedSubscribers>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
      <FailedSubscriber>
        <SourceSubscriber xsi:nil="true" />
        <ErrorMessage>string</ErrorMessage>
      </FailedSubscriber>
    </FailedSubscribers>
  </SubscribersInsertResult>
```

Common error: similar to insert and update methods.

2.2.8 Method *SubscribersExport*

Usage: Exports subscribers from system. You are able to create complex query to export only those subscribers you really need (eg those from one town, or only males, or those younger than some age etc.) If you don't specify any query, method will export all subscribers for given export type.

Address

https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersExport

Input Parameters:

- ExportType - one can choose between following exportTypes: Active, Inactive, All, Bouncers, HardBouncers, SoftBouncers or SubscriberHistory.
- SearchConditions– Parameter is an array of SearchCondition objects. One can send several SearchCondition objects within this parameter describing several attributes and create "smart" query with OR or AND attributes specified in SearchCondition->NextConditionOperator parameter.The searchCondition looks like this:

```
<s:complexType name="SearchCondition">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Conditions" type="tns:ArrayOfCondition" />
      <s:element minOccurs="1" maxOccurs="1" name="ConditionsJoining" type="tns:ConditionJoining" />
      <s:element minOccurs="1" maxOccurs="1" name="NextSearchConditionJoining" type="tns:ConditionJoining" />
```

```
        </s:sequence>
      </s:complexType>
```

ConditionsJoining is representing joining between two of the Condition elements, it can be AND or OR. Same stands for NextConditionJoining but it represents joining of this element with next SerachCondition element.
Condition class looks like this:

```
<s:complexType name="Condition">
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="Operator" type="tns:ConditionOperator" />
      <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
    </s:sequence>
  </s:complexType>
```

- Besides using subscriber attributes as search conditions, one can use several non attribute fields:
    - signintimestamp – timestamp of subscriber creation
    - signouttimestamp - timestamp of subscriber deactivation
    - lastchangetimestamp – timestamp of last subscriber data update operation
    - doiconfirmedtimestamp – timestamp of subscriber DOI activation
    - sbr_persongroupid – Id of a person group to which subscribers belong
    - sbr_targetgroupid – Id of a target group to which subscribers belong (*note – when using sbr_targetgroupid as a parameter, you can use only “Equal” Condition operator)

ConditionOperator can be one of the following:

```
<s:simpleType name="ConditionOperator">
  <s:restriction base="s:string">
    <s:enumeration value="Equal" />
    <s:enumeration value="NotEqual" />
    <s:enumeration value="BiggerThen" />
    <s:enumeration value="LowerThen" />
    <s:enumeration value="Contains" />
    <s:enumeration value="NotContains" />
    <s:enumeration value="IsNull" />
    <s:enumeration value="IsNotNull" />
  </s:restriction>
</s:simpleType>
```

Response: returns an array of SubscriberExport objects:

```
<SubscribersExportResult>
    <SubscriberExport>
      <SignIn>dateTime</SignIn>
      <LastChange>dateTime</LastChange>
      <SignOut>dateTime</SignOut>
      <IsActive>boolean</IsActive>
      <Fields>
        <Field xsi:nil="true" />
        <Field xsi:nil="true" />
      </Fields>
      <Head>string</Head>
      <Interests>
        <Interest xsi:nil="true" />
        <Interest xsi:nil="true" />
      </Interests>
      <HistoryData>
        <signInInfo xsi:nil="true" />
        <lastChange xsi:nil="true" />
        <signOutInfo xsi:nil="true" />
        <receivedNewsletters xsi:nil="true" />
        <DOIConfrimed>boolean</DOIConfrimed>
      </HistoryData>
    </SubscriberExport>
    <SubscriberExport>
      <SignIn>dateTime</SignIn>
      <LastChange>dateTime</LastChange>
      <SignOut>dateTime</SignOut>
      <IsActive>boolean</IsActive>
      <Fields>
        <Field xsi:nil="true" />
        <Field xsi:nil="true" />
      </Fields>
      <Head>string</Head>
      <Interests>
        <Interest xsi:nil="true" />
        <Interest xsi:nil="true" />
      </Interests>
      <Where>
        <Condition xsi:nil="true" />
        <Condition xsi:nil="true" />
      </Where>
      <ResetAllInterests>boolean</ResetAllInterests>
      <HistoryData>
        <signInInfo xsi:nil="true" />
        <lastChange xsi:nil="true" />
        <signOutInfo xsi:nil="true" />
        <receivedNewsletters xsi:nil="true" />
        <DOIConfrimed>boolean</DOIConfrimed>
      </HistoryData>
    </SubscriberExport>
  </SubscribersExportResult>
  <ErrorMessage>string</ErrorMessage>
</SubscribersExportResponse>
```

Common error: Error can occur usually if your condition is invalid, if you search for non-existing attributes or some other non-logical query. Next common error is when operators are used to non-belonging attributes data types (e.g. bigger or less are used for string data types).

2.2.9 Method *SubscribersExportAsync*

Usage: Asynchronously exports subscribers from system and writes them to export file. You are able to create complex query to export only those subscribers you really need (e.g. those from one town, or only males, or those younger than some age etc.) If you don't specify any query, method will export all subscribers for given export type.
Address

https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersExportAsync
Input Parameters:

- ExportType - one can choose between following exportTypes: Active, Inactive, All, Bouncers, HardBouncers or SoftBouncers.
- SearchConditions– Parameter is an array of SearchCondition objects. One can send several SearchCondition objects within this parameter describing several attributes and create "smart" query with OR or AND attributes specified in SearchCondition->NextConditionOperator parameter. ProjectId cannot be sent in search condition and it must be specified in ProjectId parameter. The searchCondition looks like this:

```
<s:complexType name="SearchCondition">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Conditions" type="tns:ArrayOfCondition" />
      <s:element minOccurs="1" maxOccurs="1" name="ConditionsJoining" type="tns:ConditionJoining" />
      <s:element minOccurs="1" maxOccurs="1" name="NextSearchConditionJoining" type="tns:ConditionJoining" />
    </s:sequence>
  </s:complexType>
```

  ConditionsJoining is representing joining between two of the Condition elements, it can be AND or OR. Same stands for NextConditionJoining but it represents joining of this element with next SerachCondition element.
  Condition class looks like this:

```
<s:complexType name="Condition">
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="Operator" type="tns:ConditionOperator" />
      <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
    </s:sequence>
  </s:complexType>
```

- Besides using subscriber attributes as search conditions, one can use several non attribute fields:
  - signintimestamp – timestamp of subscriber creation
  - signouttimestamp - timestamp of subscriber deactivation
  - lastchangetimestamp – timestamp of last subscriber data update operation
  - doiconfirmedtimestamp – timestamp of subscriber DOI activation
  - sbr_persongroupid – Id of a person group to which subscribers belong
  - sbr_targetgroupid – Id of a target group to which subscribers belong (*note – when using sbr_targetgroupid as a parameter, you can use only "Equal" Condition operator)

ConditionOperator can be one of the following:

```
<s:simpleType name="ConditionOperator">
  <s:restriction base="s:string">
    <s:enumeration value="Equal" />
    <s:enumeration value="NotEqual" />
    <s:enumeration value="BiggerThen" />
    <s:enumeration value="LowerThen" />
    <s:enumeration value="Contains" />
    <s:enumeration value="NotContains" />
```

```
        <s:enumeration value="IsNull" />
        <s:enumeration value="IsNotNull" />
      </s:restriction>
    </s:simpleType>
```

- ProjectId – Mandatory parameter that contains project id for which export is done.
- Separator – Separator that is used to split values in export file. It can be any string. Default separator is ";".

Response: returns integer value of export process id
`<SubscribersExportAsyncResult>int</SubscribersExportAsyncResult>`

Output parameters: If an error occurs in method invoke this parameter returns error message
`<ErrorMessage>string</ErrorMessage>`

Common error: Error can occur usually if your condition is invalid, if you search for non-existing attributes or some other non-logical query. Next common error is when operators are used to non-belonging attributes data types (e.g. bigger or less are used for string data types).

2.2.10 Method *SubscribersExportResult*

Usage: Returns results of SubscribersExportAsync method.

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersExportResult

Input Parameters:
- ExportId – Id of export process that is returned from SubscribersExportAsync method.
- ReturnFileIfCompleted – Boolean value indicating if the method should return export file containing subscribers if the export process is successfully completed.

Response: Returns SubscribersExportResult object

```
<SubscribersExportResultResponse xmlns="http://mission-one.de/">
   <SubscribersExportResultResult>
     <ExportId>int</ExportId>
     <ExportState>Running or ErrorOccurred or Finished</ExportState>
     <TotalSubscribers>int</TotalSubscribers>
     <Exported>int</Exported>
     <Subscribers>base64Binary</Subscribers>
     <ErrorMessage>string</ErrorMessage>
   </SubscribersExportResultResult>
 </SubscribersExportResultResponse>
```

This object contains:
- ExportId – id of export process
- ExportState – current state of export process. Possible values are: Running, ErrorOccurred, Finished

- TotalSubscribers – Total number of subscribers that will be exported in this export process
- Exported – Number of exported subscribers so far
- Subscribers – base64Binary field containing file with exported subscribers (returned only if process is finished and ReturnFileIfCompleted is true)
- ErrorMessage – String containing error message if an error occurred during export process or during retrieval of results

Common error: None

2.2.11 Method *SubscriberNewsletterSend*

Usage: Sends newsletter to subscriber provided in parameters.

Address
http://appserver.permission-one.de/SubscribersService/CustomerInterfaceV2.asmx?op=SubscriberNewsletterSend

Input Parameters:
- subscriber – subscriber object, representing subscriber we want to receive newsletter. It's enough to provide "condition" informations inside Subscriber.Where property. (check SubscribersUpdate method).
- newsletterId – representing newsletter id value for the newsletter we want to send.

Response: As response contains Boolean information whether operation succeed and message object which would contain error text if any.

```
        <SubscriberNewsletterSendResult>boolean</SubscriberNewsletterSendResult>
       <Message>string</Message>
```

Common error: Providing subscriber that does not exist, newsletter that does not exist or is not ready for dispatch.

2.2.12 Method *ScoringInterestsAction*

Usage: Insert or export interests history for provided SubscriberId or SubscriberMid.

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=ScoringInterestsAction

Input Parameters:
- Action – determines action of method. Possible values are: None, Export or Import.
- SubscriberId – id of subscriber (should be null in case SubscriberMid is used)
- SubscriberMid – subscriber's mid (should be null in case SubscriberId is used)

- Interests – input/output object, should be used as input parameter when Action is Import, contains interests separated with ‘;’
- ErrorMessage –reference, used as output parameter
- IsError –reference, used as output parameter

Response: As response is used:

- Interests – input/output object, should be used as output parameter when Action is Export, contains interests from history
- ErrorMessage – reference, used as output parameter, contains error message if there is error happened
- IsError – reference, used as output parameter, has value true in case error happened

Common error: Error can occur if incorrect interest is contained in ‘;’ separated interests in string Interests, or any other interest is mixed with interest 0 which represents all interests.

2.3 Blacklist - methods

This method is used for maintenance and administration of the internal mission<control> blacklist that is valid customer wide. All recipients whose email address is conform to one of the set patterns will not be added to the database or do not receive a newsletter from the mission<control> (in case these subscribers have already been entered). If an item has been added it will be valid until it is deactivated via web service or the mission<control>.

2.3.1 Method *BlackListPatternExist*

Usage: Checks whether blacklist pattern provided exists on the system.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=BlackListPatternExists
Input Parameters:

- Pattern – Complex object describing pattern and is used in all BlackListMethods so it is going to be described only once:
    - Description (string) – Description is considered only when you insert pattern so you can put some meaningful description here.
    - LocalPart (string) – is the part before the @ sign of the address (e.g.: marc.mustermann)
    - DomainPart (string) – is the part after the @ sign of the email address (e.g.: example.com)
    - Action (enum) – is used for processing methods, can be Insert or Delete.
    - IsForAllProjects (Boolean) – True or false, describing whether pattern is used widely on projects or not.

Response: returns Boolean BlackListPatternExistsResult (if exist, then the response is true) and ErrorMessage if something wrong occurred.

Common error:
none

2.3.2 Method *BlackListPatternsGetAll*

Usage: Returns all patterns from system.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=BlackListPatternsGetAll
Input Parameters:

- Pattern – Complex object describing pattern and is used in all BlackListMethods so it is going to be described only once:
    - Description (string) – Description is considered only when you insert pattern so you can put some meaningful description here.

- LocalPart (string) – is the part before the @ sign of the address (e.g.: marc.mustermann)
- DomainPart (string) – is the part after the @ sign of the email address (e.g.: example.com)
- Action (enum) – is used for processing methods, can be Insert or Delete.
- IsForAllProjects (Boolean) – True or false, describing whether pattern is used widely on projects or not.

Response: returns boolean BlackListPatternExistsResult (if exist, then the response is true) and ErrorMessage if something wrong occurred.

Common error:
none

2.3.3 Method *BlackListPatternsDeleteAll*

Usage: deletes all patterns from system.
Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=BlacklistPatternsDeleteAll
Input Parameters: none
Response: string error message; in case some error occurred.
Common error:
none

2.4 PersonGroup - methods

These methods are used for processing persongroups in the system. Means, it is possible to delete, edit or insert persongroup to the system.

2.4.1 Method *PersonGroupDelete*

Usage: Checks whether blacklist pattern provided exists on the system.
Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=PersonGroupDelete

Input Parameters:
- PersonGroup – Complex object describing Person Group and is used in all PersonGroup Methods so it is going to be described only once:
  - Id (int) – Representing ID of the persongroup which can be used with delete method.
  - ProjectId (int) – Representing project id of the persongroup.
  - Name (string) – Name of the persongroup.
  - Description (string) – When inserting PG you can describe it in this field.

Additional Note: When using this method, its enough to set ID or projected + Name for the PersonGroup object to have a valid PG object for deletion.
Response: returns bolean result (if deleted, then the response is true) and ErrorMessage if something wrong occurred.

Common error:
If you did not fufllfil the personGroup object correctly (check additional note) then its return "False" with errorMessage descrbing what you did wrong.

2.4.2 Method *PersonGroupGet*

Usage: Returns array of PersonGroup objects from the system with all data fullfiled (id, name, project and description).
Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=PersonGroupGet
Input Parameters:
- PersonGroup – Complex object – described in previous method.

Additional Note: When using this method, it is possible to search by:
- Person group id
- Person group name and project id
- Project id.

To search by these conditions it is necessary to fill these data in person group input parameter.

Response: returns array of PersonGroup objects with all data fullfiled.

Common error: If you did not fufllfil the personGroup object correctly (check additional note) then it returns "False" with errorMessage describing what you did wrong.

If you did not fulfill the personGroup object correctly (check additional note) then it returns "null" with errorMessage describing what you did wrong.

2.4.3 Method *PersonGroupInsert*

Usage: Returns PersonGroup object from the system with all data fullfiled (id, name, project and description).
Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=PersonGroupInsert
Input Parameters:

- PersonGroup – Complex object – described in previous method.

Additional Note: When using this method, its mandatory to set projectId and Name of the persongroup. Method will return object which is inserted, means it will return ID of newly created PersonGroup.

Response: returns PersonGroup object with all data fullfiled.

Common error: Don't forget to set the Name and Project for PersonGroup you want to insert.

If you did not fufllfil the personGroup object correctly (check additional note) then its return "null" with errorMessage descrbing what you did wrong.

2.4.4 Method *PersonGroupsGetByDispachedDaysAgo*

Usage: This is special method requested by some customers. It returns PersonGroup objects list from the system with all data fullfiled (id, name, project and description), same as PersongroupsGet method, but this method returns only those that are dispatched before current date subtracted by "days ago" parameter value provided. So this means, if one sends "daysAgo" parameter with value 5, then method will return all dispatched persongroups from beginning until five days ago.

Address:
http://appserver.permission-one.de/SubscribersService/CustomerInterfaceV2.asmx?op=PersonGroupsGetByDispachedDaysAgo

Input Parameters:

- DaysAgo – integer – Described in usage.

Response: returns PersonGroup objects with all data fullfiled like other persongroups methods.

Common error: none.

2.6 Other Methods
Here will be described other methods that are not belonging to any other group.

2.6.1 Method *ShopTracker*

Usage: This method is used in combination with Customer's web-shop. Main usage is to insert web-shop return information for newsletter buying's so one can track incomes for dispatches made. Each order made via newsletter should be inserted to the system using this Web-service method. One order, one method call.

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=ShopTracker
Input Parameters:

- Order– Complex object representing order with details. Order object is containing:
    - Mid String that you get from your newsletter cliks within GET parameter of the url (see other methods using this parameter)
    - OrderDate – Date when the order have been made.
    - ItemCount – Number of items been sold within this order.
    - OrderValue – Total Value of an order in eurocents.
    - OrderItems – Additionally, each order item can be described in details with this complex object.
    - Properties – Used for OrderItems – contains KeyID and ParentKeyID.

Response: Returns ShopTrackerResult as Boolean and ErrorMessage if something wrong occurs.
Common error:
Invalid MID sent to the methods usually cause most of the problems.

2.6.2 Method *InterestExport*

Usage: Method used for exporting interests. Method is use-full to see interest ID mapping so you can insert interests or change them to your subscribers.

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=InterestsExport
Input Parameters: bi-directional ErrorMessage parameter (string).

Response: complex object with error message input parameter. Object contains array of CustomerInterest object with Name and Id values.

```
    <InterestsExportResponse>
      <InterestsExportResult>
        <CustomerInterest>
          <Value>int</Value>
          <InterestName>string</InterestName>
        </CustomerInterest>
        <CustomerInterest>
          <Value>int</Value>
          <InterestName>string</InterestName>
        </CustomerInterest>
      </InterestsExportResult>
      <ErrorMessage>string</ErrorMessage>
    </InterestsExportResponse>
```

Common error:
None.

2.6.3 Method *NewsletterKeyGet*

Usage: Method returns Newsletter Key which can be used with other methods. There are three input parameters where at least one has to be sent. If more is sent, only one is considered (Mid, then newsletter, then dispatchpartid).

Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=NewsletterKeyGet

Input Parameters:

- Mid – This is known and already explained parameter.
- NewsletterId – Id of the newsletter that you want key to get.
- DispatchPartId – Get the key of the newsletter used for this dispatch part.

Response: String – NewsletterKey and bi-directional error message.
Common error:
Invalid MID sent to the methods usually cause most of the problems. None of the parameter is sent.

2.6.4 Method *NewsletterArchiveUrlGet*

Usage: Method returns NewsletterArchive list of objects which contains all basic information for newsletter including newsletter Url.

Address:
http://appserver.permission-one.de/SubscribersService/CustomerInterfaceV2.asmx?op=NewsletterArchiveUrlGet

Input Parameters:

- newsletterId– Array of newsletter related to those one want to get archive newsletter object.
- projectId– Project id, means it will return all newsletters data for provided object.

Response:
```
<NewsletterArchiveUrlGetResult>
        <NewsletterArchive>
          <NewsletterSubject>string</NewsletterSubject>
          <NewsletterUrl>string</NewsletterUrl>
          <LastChangeDate>string</LastChangeDate>
          <NewsletterId>int</NewsletterId>
        </NewsletterArchive>
        <NewsletterArchive>
          <NewsletterSubject>string</NewsletterSubject>
          <NewsletterUrl>string</NewsletterUrl>
          <LastChangeDate>string</LastChangeDate>
          <NewsletterId>int</NewsletterId>
        </NewsletterArchive>
      </NewsletterArchiveUrlGetResult>
      <message>string</message>
```
Common error:
- None of input parameters are not sent. (this would produce error call).
- Trying to get too much data at once by sending to big newsletterId array or providing project with tons of newsletters in the project. This can possibly produce timeout or too slow response from server

2.7 Statistics Methods
There are two statistics methods until the date. They are used to process results of dispatches and show their response.

2.7.1 Method *DispatchesGet*

Usage: Method is used for different queries to dispatch statistics. Depending on input parameters it can calculate:
- All dispatches statistics
- Dispatches for given date period
- All dispatches for one project id
- All dispatches for one newsletter id
- Dispatch statistics for one dispatch or dispatch part (In this case method returns array of one, or none if no dispatch exists, DispatchStatistics class object)

Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=DispatchesGet

Input Parameters:

- startDate – DateTime null able parameter (described below)
- endDate – DateTime null able parameter (described below)
- projectId – projectId, described below.
- newsletterId –described below.
- dispatchId – Id of the dispatch, described below.
- dispatchPartId – Id of the dispatch part, described below.

Method input parameters are checked from first to last parameter. If first parameters, startDate and endDate, are provided (not null) method will return dispatch statistics for given period and other parameters are discarded (So if Customer sends startDate, endDate and projectId, projectId parameter is discarded).
If all input method parameters are null (not provided) method returns all dispatches statistics.
If startDate and endDate parameters are provided method returns dispatches statistics for given period. If one of these parameters is null method will not return statistics for period, and instead it will check for other parameters (projectId, newsletterId ...).
If projectId parameter is provided method will return all dispatches statistics for this project id (Request 3).
If newsletterId parameter is provided method will return all dispatches statistics for this newsletter id.
If dispatchId parameter is provided method will return dispatch statistics for this dispatch id. If this dispatch exists in database method will return array of one element, if it does not exists method will return empty array.
If dispatchPartId parameter is provided method will return dispatch statistics for this dispatch part id. If this dispatch part exists in database method will return array of one element, if it does not exist method will return empty array.

Response: Response is represented by complex DispatchStatistics object array. Fields you can find complex object are:

```
 DispatchStatus  {      Running,       Finished,       Aborted    }
DispatchID
NewsletterID
IsTest  -Test or production newsletter
DispatchStatus Status  //complex
SentMailsHtml //count of
SentMailsText //count of
BouncersHard  //count of
BouncersSoft //count of
ClicksNeto  //count of
ClicksBruto //count of
Openings //count of
DispatchStartTimestamp
NewsletterUrlStatistics[] NewsletterUrls  // contains Url, clicksNeto and clicksBrutto.
```

Class field „NewsletterUrlStatistics[] NewsletterUrls“ is returned only for one request (dispatch statistics for given dispatchId or dispatchPartId). It contains all urls for that newsletter with clicks count.

ErrorMessage out parameter is empty string if method is completed successfully. If some kind of error occurs, this parameter returns error message.

Common error:

Invalid input parameters.

2.7.2 Method *DispatchesExport*

Usage: Method is exporting subscriber but only for certain dispatches. This method is used for two statistic exports: 1.All persons that clicked or opened certain newsletter 2.All subscribers for which newsletter is sent

Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=DispatchesExport

Input Parameters:
DispatchExportType dispatchExportType – this parameter is mandatory and has two possible values: DispatchReport, NewsletterExport. If parameter value is NewsletterExport method returns all persons that clicked or opened certain newsletter. If parameter value is DispatchReport method returns all subscribers for which newsletter is sent.
Parameters dispatchId and dispatchPartId are optional. If one of these parameters is provided, method will return statistics for given parameter. If parameter dispatchId is provided, parameter dispatchPartId is discarded.
Parameter string[] subscriberFields – is optional. If this parameter is provided and dispatchExportType is NewsletterExport method will return only subscriber fields that are contained in this input array. If dispatchExportType is DispatchReport, this parameter is discarded.

Response: Its an array of subscriberExportDispatch objects. This object is containing all fields like normal Subscriber object (see previous methods), with addition for following parameters:
 SubscriberDispatchType // can be: 0 Successful, 1 – Soft bouncer, 2 – Hard bouncer
 DispatchId
 Timestamp
 string URL //
This parameter is returned only if dispatchExportType is DispatchReport.

Class fields DispatchId, Timestamp and URL are returned only if dispatchExportType is NewsletterExport.
URL field can have two values: “URL” or “OPEN”.

ErrorMessage out parameter is empty string if method is completed successfully. If some kind of error occurs, this parameter returns error message.

Common error:
Invalid parameters.

2.7.3 Method *ProjectsGet*

Usage:
Returns array of *ProjectExport* objects from the system with all project data fulfiled.

Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=ProjectsGet

Input parameters:

- ProjectId (Optional) – id of the project
- ProjectName (Optional) – name of the project
- TagsSearchCondition – Complex object describing project tags to be used as a search condtion. One can send several ProjectTagsSearchCondition objects within this parameter and create “smart” query.The ProjectTagsSearchCondition looks like this:

```
<s:complexType name="ProjectTagsSearchCondition">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="TagConditions"
type="tns:ArrayOfProjectTagCondition" />
        <s:element minOccurs="1" maxOccurs="1" name="ConditionsJoining"
type="tns:ConditionJoining" />
        <s:element minOccurs="1" maxOccurs="1"
name="NextSearchConditionJoining" type="tns:ConditionJoining" />
    </s:sequence>
</s:complexType>
```

ConditionsJoining is representing joining between two of the *TagConditions* elements, it can be *AND* or *OR*. Same stands for *NextSearchConditionJoining*, but it represents joining of this element with next *ProjectTagsSearchCondition* element.
TagConditions class looks like this:

```
<s:complexType name="ProjectTagCondition">
    <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="Operator"
type="tns:ConditionOperator" />
        <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string" />
    </s:sequence>
</s:complexType>
```

ConditionOperator can be one of the following:

```
<s:simpleType name="ConditionOperator">
  <s:restriction base="s:string">
    <s:enumeration value="Equal" />
    <s:enumeration value="NotEqual" />
    <s:enumeration value="BiggerThen" />
    <s:enumeration value="LowerThen" />
```

```
      <s:enumeration value="Contains" />
      <s:enumeration value="NotContains" />
      <s:enumeration value="IsNull" />
      <s:enumeration value="IsNotNull" />
    </s:restriction>
  </s:simpleType>
```

BiggerThen and *LowerThen* cannot be used in this condition, and application will report error if you try to use it because these operators are not applicable to string objects.

Example: Here is an example of *ProjectTagsSearchCondition* array which produces tags condition like *((tags contains A3 Or tags contains A2) And (tags contains A1))*.

```
<mis:TagsSearchCondition>
        <mis:ProjectTagsSearchCondition>
                <mis:TagConditions>
                        <mis:ProjectTagCondition>
                                <mis:Operator>Contains</mis:Operator>
                                <mis:Value>A3</mis:Value>
                        </mis:ProjectTagCondition>
                        <mis:ProjectTagCondition>
                                <mis:Operator>Contains</mis:Operator>
                                <mis:Value>A2</mis:Value>
                        </mis:ProjectTagCondition>
                </mis:TagConditions>
                <mis:ConditionsJoining>Or</mis:ConditionsJoining>
                <mis:NextSearchConditionJoining>And</mis:NextSearchConditionJoining>
        </mis:ProjectTagsSearchCondition>
        <mis:ProjectTagsSearchCondition>
                <mis:TagConditions>
                        <mis:ProjectTagCondition>
                                <mis:Operator>Contains</mis:Operator>
                                <mis:Value>A1</mis:Value>
                        </mis:ProjectTagCondition>
                </mis:TagConditions>
                <mis:ConditionsJoining>Or</mis:ConditionsJoining>
                <mis:NextSearchConditionJoining>And</mis:NextSearchConditionJoining>
        </mis:ProjectTagsSearchCondition>
</mis:TagsSearchCondition>
```

Additional Note: *If all search parameters (project id, project name and tags) are null method will return all projects.*

Response: Array of *ProjectExport* objects.

2.8 Subscribers import methods

2.8.1. Method *SubscribersImportAsync*

Usage: This method is used to asynchronously start the import process via CI-2 and to check the results later while the import is running in background. The import uses "Fast import" option when invoking the import service.
Address:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersImportAsync

Input parameters:
- Subscribers – array of Subscribers. This parameter is used if user sent null as ImportFile parameter. When using array of subscribers as import source, this method will process only fields and interest from each element in subscriber array. It is not required that all subscribers have same fields and in same order. Fields which are found in one subscriber and not in some other subscribers will be set to empty value for subscriber that is missing these values.
- ImportFile – array of bytes. This parameter is used to send file to asynchronous import process. It is possible to send text file or zipped file. When sending this parameter it is required to send FileDescription parameter.
- FileDescription – description of ImportFile parameter. If user sent array of subscribers as import source than this parameter is ignored. It contains three fields:
    - AttributePairings – array of FileAttributePairing. Pairings of columns in file with attributes in database. It contains two strings FileColumnName and AttributeName. This parameter is required to bind columns from import file with subscriber attributes.
    - Separator – separator used to separate values and column names in import file. Possible values are None, Tab, Semicolon, Comma and Space. User has to provide one separator. If separator is none, method will return error message.
    - FileType – type of import file. Possible values are None, Text and Zip. If value None is sent, method will try to parse it as a text file.
- ImportSettings – various setting important for import process. This parameter is required. It contains following fields:
    - ImportType – type of import process. Possible values are: None, Import, ImportOrUpdate and Update. None uses default value, Import is used for inserting subscribers, ImportOrUpdate for updating subscribers and insert if not found and Update is used to update subscribers.
    - InterestParameter – used for subscriber interests. Possible values are: None, Append and OverWrite.

- UniqueAttributeId – id of unique attribute used when updating subscribers. This parameter must be present in import file (through FileAttributePairing) or in subscriber fields. Otherwise, method will return error message.
- ProjectIds – array of project ids. There must be at least one project id in this array or the method will return error message.
- PersonGroup2Subscriber – array of person groups. This method will create person groups that doesn't exist in database.
- AfterInsertAction – used for sending newsletter. Possible values are: DoiNewsletter, SigninNewsletter and None.
- ManualNewsletterId – optional Newsletter id if user selected sending newsletters.

Output parameters:
ErrorMessage – If an error occurs value of this parameter will be set to error message.
InfoMessage – Not used.

Return values:
This method returns import process id if method finished successfully, and null if method failed.

Common error:
When using package input, customers usually send projected within subscribers object. That's not usage here, as all subscribers are inserted into same project chosen in other settings.

2.8.2 Method *SubscribersImportResult*

Usage
This method is used to get current state of import such as currently imported subscribers count, currently failed subscribers count, total subscribers count… and in the end, when import is finished, to download import file with errors or to download package with failed subscribers.

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersImportResult

Input parameters:
- ImportProcessId – id of import process in the database (this parameter is returned from SubscribersImportAsync method).
- ResponseDataType ResponseDataType – type of response package containing errors if any occurred. Possible values are: None, File and Package.

Return value:
- ImportResult – object containing information about this import process.

- ImportState – State of import process. Possible values are: Invoking, Running, Paused, PendingResume, PendingPause, PendingStop, StoppedManually, ErrorOccurred and Finished,
- ErrorMessage - Determine what happened with your import,
- CountCurrentSucceded - Current number of successfully imported subscribers,
- CountCurrentFailed - Current number of failed subscribers,
- CountTotal - Total number of subscribers sent to import,
- ErrorFile –An error file to download. It is possible to download it only if invoking method with ResponseDataType.File, and only when import is finished,
- FailedSubscribers - Failed subscribers array. It will have value only if invoking method with ResponseDataType.Package and only if import is finished.

Common error:
None

2.8.3 Method *SubscribersImportAction*

Usage
This method is used to send command to asynchronous import process (pause, stop, resume).

Address
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx?op=SubscribersImportAction

Input parameters:
- ImportProcessId – id of import process in database.
- ImportCommand – command that is sent to asynchronous import process. Possible values are: None, Pause, Resume and Stop. It is possible to pause and stop only running processes. Resume is only possible for paused process.
- ErrorMessage – parameter containing error message if any error occurred during sending command.

Return values:
- Bool – indication if command is sent successfully or not.

Common error:
none

3. Access to the mission<one>control API

3.1 API URI – Referencing the web service

You get access to the web service via internet using the following URL:
https://appserver.permission-one.de/subscribersservice/customerinterfacev2.asmx

Access to the methods of the web service can be implemented into existing systems easily. A simple integration is processable with one of the following programs:

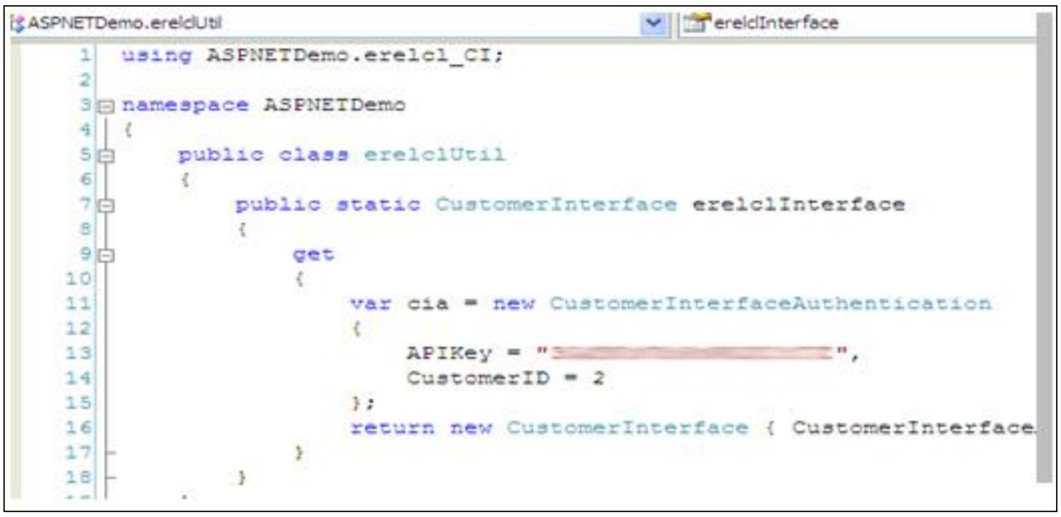
- Microsoft.NET
- PHP
- SOAP for Python
- SOAP for Ruby
- JAVA
- And many other program languages

mission<one> offers demo applications for PHP and ASP.NET (C#). These demos explain the easy connection to the web service and can be used as instructions for system developer. You find detailed examples in the according documents and program codes.

3.2 Access to API

The mission<cl> CustomerInterface API is protected against unauthorized access by an authentication mechanism. If you want to exchange data with the mission<cl> using the API you need a customerID and the appropriate APIkey.

You get these data for accessing the API from your contact at mission<one>.



```
using ASPNETDemo.erelcl_CI;

namespace ASPNETDemo
{
    public class erelclUtil
    {
        public static CustomerInterface erelclInterface
        {
            get
            {
                var cia = new CustomerInterfaceAuthentication
                {
                    APIKey = "[redacted]",
                    CustomerID = 2
                };
                return new CustomerInterface { CustomerInterface
            }
        }
```

image 1: Authentication

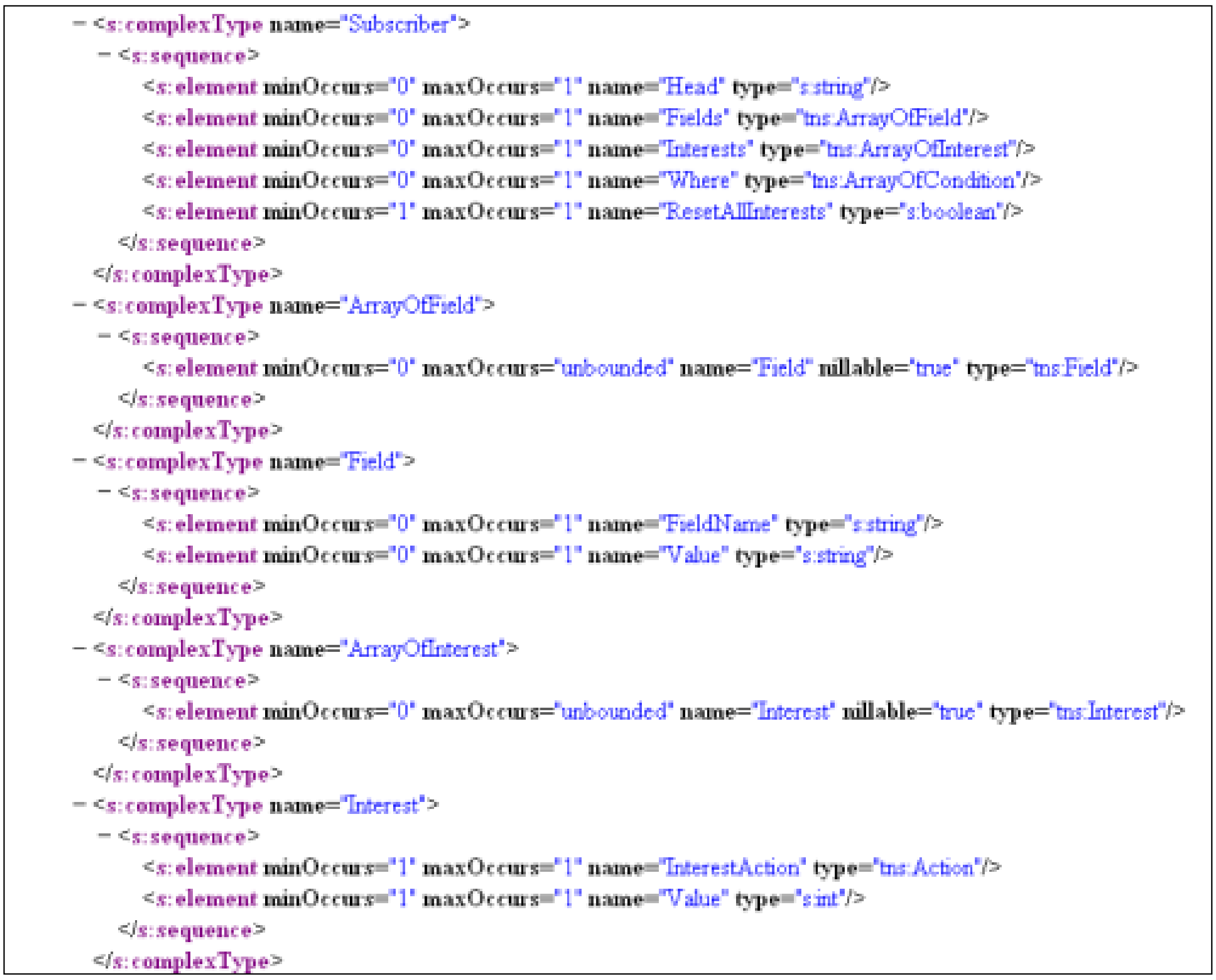
After successful registration you can use these methods. The authentication has to be processed for one session only once.

3.3 Subscriber Object

A subscriber object presents a subscriber. An object 'subscriber' has the following attributes:

- Fields: Attributes the subscriber database
- Interests: Interests of the subscribers
- Head: detailed information of the client (optional)
- Condition: Condition parameter so called WHERE Statements that is important for selecting, deleting and updating of subscriber data.

The following image shows the object type 'Subscriber' with all characteristics.



```
- <s:complexType name="Subscriber">
  - <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Head" type="s:string"/>
      <s:element minOccurs="0" maxOccurs="1" name="Fields" type="tns:ArrayOfField"/>
      <s:element minOccurs="0" maxOccurs="1" name="Interests" type="tns:ArrayOfInterest"/>
      <s:element minOccurs="0" maxOccurs="1" name="Where" type="tns:ArrayOfCondition"/>
      <s:element minOccurs="1" maxOccurs="1" name="ResetAllInterests" type="s:boolean"/>
    </s:sequence>
  </s:complexType>
- <s:complexType name="ArrayOfField">
  - <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Field" nillable="true" type="tns:Field"/>
    </s:sequence>
  </s:complexType>
- <s:complexType name="Field">
  - <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="FieldName" type="s:string"/>
      <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:string"/>
    </s:sequence>
  </s:complexType>
- <s:complexType name="ArrayOfInterest">
  - <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="Interest" nillable="true" type="tns:Interest"/>
    </s:sequence>
  </s:complexType>
- <s:complexType name="Interest">
  - <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="InterestAction" type="tns:Action"/>
      <s:element minOccurs="1" maxOccurs="1" name="Value" type="s:int"/>
    </s:sequence>
  </s:complexType>
```

Image 2: Object-Type Subscriber

The characteristics are described in detail as follows.

3.3.1 Fields

Fields are a list of attributes/fields of subscribers. An attribute is always presented by a field name and the according value.

3.3.2 Interests

Interests are the fields of interest of a subscriber. It can be ignored if interests are not relevant. Detailed information for using the fields of interests is available from the mission<control> manual. Interests are mandatory when inserting, see the demos.

3.3.3 Head

Clients using the web service can enter their own information in the attribute 'Head'. This could be e.g. an ID from their own system for identification of the transmitted data. In case of failure the web service will return these data as well.

3.3.4 Condition

Conditions are term parameters that are important for updating and deleting subscriber data. One condition has to be given for these processes. There are two possibilities:

1. Client indicates an UNIQUEfield (e.g. CustomerID from the own CRM system that is available in the mission<CL>)
2. Client indicates the email address and the ProjectID

According to these conditions the subscriber can be identified and processed. When entering new subscribers such conditions are not important.

After setting the subscriber objects you can use these methods for transmitting data (detailed information under Demo Program Codes). Alternatively you can also transmit an XML packet as described as following.

3.4 Subscriber XML Packet

Hereinafter it is shown what an XML packet looks like that is transmitted to the web service. This packet reflects a serialized object of a subscriber type.

```
<SubscribersPackage>
<Subscribers>
 <Subscriber>
        <Head>Update Test HEAD</Head>
        <Fields>
                <Field>
                        <FieldName>vorname</FieldName>
                        <Value>Edin2</Value>
                </Field>
                <Field>
                        <FieldName>nachname</FieldName>
                        <Value>Kadic</Value>
                </Field>
        </Fields>
        <MultiSelectFields>
         <MultiSelectField>
           <FieldName>first fieldname</FieldName>
                <Values>
                     <FieldValue> <Value>4</Value> </FieldValue>
                     <FieldValue> <Value>6</Value> </FieldValue>
                     <FieldValue> <Value>9</Value> </FieldValue>
                </Values>
            <InsertValueIfNotExists>false</InsertValueIfNotExists>
        </MultiSelectField>
        <MultiSelectField>
           <FieldName>second fieldname</FieldName>
                  <Values>
                     <FieldValue> <Value>4</Value> </FieldValue>
                     <FieldValue> <Value>7</Value> </FieldValue>
                  </Values>
           <InsertValueIfNotExists>true</InsertValueIfNotExists>
         </MultiSelectField>
    </MultiSelectFields>
        <Interests>
                <Interest>
                        <InterestAction>Delete</InterestAction>
```

```
                    <Value>0</Value>
                    </Interest>
                    <Interest>
                              <InterestAction>Insert</InterestAction>
                    <Value>1</Value>
                    </Interest>
          </Interests>
          <Where>
                    <Condition>
                              <Operator>Equal</Operator>
                    <FieldName>email</FieldName>
                          <Value>missionone1@lycos.de</Value>
                    </Condition>
                    <Condition>
                              <Operator>Equal</Operator>
                  <FieldName>prj_projectid</FieldName>
                              <Value>1</Value>
                    </Condition>
          </Where>
   </Subscriber>
 </Subscriber>
</SubscribersPackage>
```

The answer of the web service that is returned after deletion, insertion or update looks as following:

```
<SubscribersServiceResponse>
          <IsError>true</IsError>
          <ErrorMessage>ERROR_BLA_BLA</ErrorMessage>
          <SucceededSubscribersCount>1</SucceededSubscribersCount>        <FailedSubscribers>
                    <FailedSubscriber>
                              <SourceSubscriber>
                              <!– Here goes original subscriber packet –>
                    </SourceSubscriber>
                              <ErrorMessage>
                          XXXX; Fault Condition Array or missing Condition Array
               </ErrorMessage>
                    </FailedSubscriber>
                    <FailedSubscriber>
                              <SourceSubscriber>
                                        <!– Here goes original subscriber packet –>
</SourceSubscriber>
         <ErrorMessage>
```

```
            XXXX; Fault Condition Array or missing Condition Array
        </ErrorMessage>
                    </FailedSubscriber>
          </FailedSubscribers>
</SubscribersServiceResponse>
```

4. API possible failure messages - Fehlermeldungen

Failure code:	Failure text	description
8100	Mandatory field not given.	The mandatory field within the subscriber packet is missing
8200	Value '{1}' is none of the single-select values [{2}	Sie versuchen nicht vorhandene Werte in ein Auswahl-Feld einzufügen
8200	Value '{1}' is none of the Multi-select values [{2}	You attempt to enter none available values into a multiple choice field.
8300	Value '{1}' is not {2}	You attempt to enter data of the wrong type, e.g. letter instead of integer
8400	Value '{1}' is not within the range from {2} to {3}	Value is not within the correct range
8500	Value '{1}' does not match regular expression '{2}'	Value does not match the regular expression
8600	Value '{1}' exceeds the maximum length of {2}	Value you want to enter is too long
9001	Person group with id {0} does not exist	This person group is not available
880001001	DoubleEmail	This email address already exists in this project
9002	Unexisting MultiSelect Value: {0}	Multiple choice field is not available
8801	Interests are empty	No interests are indicated. If the subscriber data do not consist of any

		interests you have to transmit at least 0.
8700	Interests;Wrong InterestID - {0}.	Transmitted interest is not available
xxxx	unexpected error	Error is not definable. Please contact the support team
400x	Fault Condition Array or missing Condition Array	WHERE Condition has not been indicated for Update or Delete
9005	Provided interest for deletion does not exists: '{0}'.	The interest for deletion does not exist.
9006	Provided interest for insertion already exists: '{0}'	The interest entered is already available on the subscriber data.
9007	You must not delete all existing interests on subscriber	You attempted to delete all interests on the subscriber.
9009	One or more mandatory fields are empty	One or more mandatory field are empty
9010	Subscriber does not exist!	Subscriber does not exist (for deletion or update)
880001002	BlackList	The email address is on the blacklist
9011	Email Address Not Provided	You missed to indicate the email address
9901	Authentication Error	There was an error on registration for the web service
9989	Invalid or not supported XML format	Error in XML Packet
9919	Server Error	Please contact our support team
9100	MultiSelectField sent Error	
	Error sending newsletter to	There is no welcome DOI newsletter set

	subscriber; settings	

5. FAQ – Frequently Asked Questions

5.1 Login Questions

F: Frequently I receive "Authentication Error" messages (Exception Message), but I definitely followed the instructions?

A: Mainly this problem occurs when entering wrong authentication data. Your CustomerID and API KeyCombination is simply wrong. Verify your data and contact the support team at mission<one>.

F: My client is not able to connect with the Customer Interface. I always receive an exception, what am I doing wrong?

A: In this case often the header information is inaccurate. Please view the demos we provided you with and deduce the correct process how to transmit the header information. Additionally you authentication information have to be correct. (see question above)

5.2 Questions to Methods

F: I always transmit the subscriber packet to all methods (Insert, Update und Delete). What is the difference between these packets?

A: Insert does not need the „WHERE part" in the subscriber packet. Update and Delete Methods need the "WHERE part" for indentifying the subscriber to be deleted or updated. Insert additionally needs all mandatory fields of the packet (e.g. „email", „prj_projectid" etc). The Delete packet needs apart from the „WHERE part" no other data.

F: When updating a subscriber I always get an error message incorrect conditions.

A: Conditions are correct if it consists of at least one unique field or with the combination of „prj_projectid" and „email".

F: Why do I get failure messages about incorrect fields?

A: Please read the SubscriberErrorMessage carefully. It indicates the errors. You have to concentrate on the following things:

- The field has to exist on the database
- The field has to be conform to the data type
- The field has to be conform to the regular expressions (in case there are any indicated, e.g. „email")
- The field has to be conform to the defined interval (if there is a definition)
- aso.

F: When inserting subscribers I get the error message: „8100: mandatory field not given"

A: Verify the mandatory fields in your system. You cannot enter subscriber data into a subscriber packet without these mandatory fields. The Subscriber.ErrorMessage indicates the involved fields.

F: When inserting subscribers I get the error message: „Error sending newsletter to subscriber; Please check your 'sign-in and DOI newsletter' settings"

A: Verify your settings of the project in the mission<control>. Usually there are no rules for welcome and DOI newsletters (double-opt-in) given. Optionally you can adjust the parameter isSignInNewsletter and isDoubleOptIn to false. Subsequent there will be no newsletter dispatched after the registration.

5.3 Questions on Interests

F: Why do I get the failure message that the interests already exist?

A: This means you attempt to allocate interests to the subscriber that already exist on its data. Use the ResetAllInterests property to delete all existing interests.

F: Why do I get the following error message „interests are empty"?

A: When inserting a subscriber the interests are a mandatory field. You have to insert a value. When updating this field it is not mandatory though. If the subscriber has no defined interest you have to indicate ‚0' as value.

F: Why do I get the following error message: „all interests deleted"?

A: You have to keep at least one field of interest for a subscriber.

F: Why do I get the following error message: „wrong interest.."?

A: The value you indicate as parameter does not exist on the system.

5.4 Questions to errors concerning email addresses

F: Why do I get the following error message: „DoubleEmail error"?

A: You attempt to import addresses that already exist on the system. An email address can only exist once for the project. Otherwise you have to accept duplications on the software for that project.

F: Why do I get the following error message: „BlackList error"?

A: You attempt to import subscribers with email addresses that exist on the blacklist.

F: Why do I get the following error message: „subscriber does not exists error"?

A: You attempt to delete or deactivate a subscriber that does not exist on the database.

5.5 AOB

F: Why do I get the following error message: „Unexpected error"?

A: This error occurs if you try to transmit a completely wrong packet (e.g. logical errors or not initialized arrays). Otherwise it is possible that our system is offline temporarily. In this case please contact the support team at mission<one>.

6. Your Contact

If you have any additional questions regarding the interfaces please do not hesitate to contact the support team at mission<one>. We were looking forward to helping you.