

# MultiEstate Export Interface specification

## Index

1. Introduction.....	2
1.1 Access to the interface.....	2
2. Configuration of the interface.....	2
2.1 Description of the configuration structure.....	2
version.....	3
offer.....	3
codebook.....	4
brokers.....	4
photos.....	4
2.2 DTD specification.....	5
3. Methods.....	5
3.1 Data types.....	6
3.2 Structure of the returned value.....	6
3.3 Compulsory methods.....	7
check.....	7
getConfig.....	8
sendOffer.....	8
deleteOffer.....	9
reportError.....	9
3.4 Optional methods.....	10
checkBroker.....	10
sendBroker.....	10
sendFreeItems.....	11
checkPhotos.....	12
sendPhoto.....	12
deletePhoto.....	13
sendCodeBook.....	13

## 1. Introduction

This document describes the principle of the client part of the export interface, which allows sending selected data from application MultiEstate to another server for its publishing or some other processing. XML-RPC is used for the data transfer, where with HTTP protocol a XML file is transferred, which contains required data in specific structure defined by XML-RPC standard. For the basic functioning a web server and whichever method of processing of received data (PHP, ASP.NET, Java...) should be sufficient. All the communication with the interface is carried out in UTF-8 coding.

### 1.1 Access to the interface

Every server requesting export of data from the application has to have a profile in the source application established, which is identified by an unique authorization code. To insure secure transfer, this key is transferred between the input parameters of the most of the methods. This can be obtained from the operator of the particular installation of MultiEstate.

## 2. Configuration of the interface

Interface is build with the goal to be to the maximum universal and reusable for whatever number of target servers. For that reason and because of the minimalization of transferred data, every profile can be configured to transfer only required data.

Basic data of the offer, like for example title, description or price (detailed overview is to be found in tables further in the text), are always sent and it's not possible to change them in the configuration. Through configuration you can designate, if your server needs data about agents, photos and other additional items of the offer. With photos you can set their size, maximum quantity and watermark, which is automatically added to these photos to unburden your server.

### 2.1 Description of the configuration structure

Complete configuration is written in the form of valid XML version 1.0, which has to comply with DTD specification described in section 2.2. This configuration has to be available to the server at all times, best to be written in a file for easy tracking of the changes.

Following is the default configuration, which will be used in case you won't provide your own configuration (output value of the ExportTime check method will be 0).

**Default export configuration**

```

<?xml version="1.0" encoding="utf-8"?>
<econfig>
  <version>1</version>
  <offer>
    <public_url />
    <freeitems enabled="true" />
  </offer>
  <codebook enabled="true">
    <class>1</class>
    <trans>1</trans>
    <stage>1</stage>
    <priceunit>1</priceunit>
    <currency>1</currency>
    <contract>1</contract>
    <freeitem>1</freeitem>
    <foptions>1</foptions>
  </codebook>
  <brokers enabled="true">
    <photos enabled="true" size="original" />
  </brokers>
  <pictures enabled="true" maxcount="0">
    <watermark>...binary data of the watermark coded in
Base64...</watermark>
    <picture size="640x480" type="jpeg" usewatermark="false" />
  </pictures>
</econfig>

```

**version**

In this branch you can determine, which version of the interface you want to use. In case there will be a new version of the interface published, you will be informed by the check method (see below). Then you can execute update of your interface according to the new specification and with the help of this configuration setting you can declare, that this new version should be used for your server. Don't declare in the configuration the version number, for which your server isn't ready. If you'll specify none or non-existing version number, the last available version will be automatically used.

**offer**

The most complicated branch **offer** represents setting of other data, which you want to get from the application additionally, and also some other settings.

First setting is the item **public\_url**, which can contain public URL address of the exported offer. This address is subsequently available for agents in the application (e.g. for the manual export verification). This value has to contain the string "#OfferId#", which will be subsequently replaced by a numeric Id obtained by the method `sendOffer`.

By setting an attribute *enabled* of the branch **freeitems**, you'll enable the use of the method for sending free items (`sendFreeItems`). In the future versions of the interface there will probably be a more detailed expansion for determining, which free items should be exported. Till then all the free items

available for the given offer will be sent.

---

## codebook

---

This branch affects the use of so-called CodeBook. Those are simple tables, where a certain verbal designation is replaced by a unique identification for simpler modification of the application (sometimes these indexes are called codebooks). Identifiers will be numeric in the most cases, but they can also be string.

Thanks to this mechanism you can assign your own verbal designation to the given identifier and/or simplify the query. If the use of the CodeBook is forbidden by this setting (attribute *enabled*), the text formulations from these tables will be sent straight instead of the identifier.

In this branch you can also assign, which specific tables you want to use, by adding another sub-branch with a name of the given table and again with the attribute *enabled*. In the default setting all the tables are set to enabled.

These tables aren't a part of this document, because they can vary and change for each specific installation of the application. Whole tables (or CodeBook) can be requested with the method "check". It is not advised to request sending of the tables at each import, because it could cause an unnecessary transfer overload. It is possible, that in some of the future versions, this behavior will be changed and the CodeBook will be sent automatically only at a modification.

---

## brokers

---

Branch **brokers** contains the attribute *enabled*, which determines, if the methods for the work with the list of agents will be called (checkBroker, sendBroker). If it is enabled, then attribute *enabled* in the branch **photos** determines, if the photos of the agents will be sent. With these photos it is possible to set, if they should be left in the original form (proportions of the sides like in a passport photo) or you can set your own dimension in the format - width x height (e.g. 20X50).

---

## photos

---

Next is the branch **photos**, with which you enable (again attribute *enabled*) sending of the photos (methods checkPhotos, sendPhoto, deletePhoto), if actually any exist with the offer. With the attribute *maxcount* you can limit the number of sent photos, this will respect the order set by the agent and the other photos will be ignored.

Inside of the branch you will find space for sending picture, which will be added into the photo as a **watermark**. For the transparent background use color #F0F0F0 (RGB: 127,127,127). The size of the picture shouldn't exceed size

640x480. Picture will be placed in the bottom center (this will be settable in one of the future versions). When no watermark will be assigned, the default logo of the application MultiEstate will be used for the requested photos.

Further with the help of particular **picture** branches you can set different formats for each picture, which should be sent. Maximum is three different formats. At the moment you can affect the size (attribute *size*), which can not be larger than 640x480, use of the watermark (attribute *watermark*) and graphic format of the final picture (attribute *type*). Values *jpeg*, *gif*, *png* are allowed. Default and recommended is the format JPEG.

## 2.2 DTD specification

```
<!DOCTYPE econfig [
<!ELEMENT version (#PCDATA)>
<!ELEMENT public_url (#PCDATA)>
<!ELEMENT freeitems EMPTY>
<!ATTLIST freeitems enabled (true|false) #REQUIRED>
<!ELEMENT offer (public_url?,freeitems?)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT trans (#PCDATA)>
<!ELEMENT stage (#PCDATA)>
<!ELEMENT priceunit (#PCDATA)>
<!ELEMENT currency (#PCDATA)>
<!ELEMENT contract (#PCDATA)>
<!ELEMENT freeitem (#PCDATA)>
<!ELEMENT foptions (#PCDATA)>
<!ELEMENT codebook
(class?,trans?,stage?,priceunit?,currency?,contract?,freeitem?,foptions?)>
<!ATTLIST codebook enabled (true|false) #REQUIRED>
<!ELEMENT photos EMPTY>
<!ATTLIST photos
  enabled (true|false) #REQUIRED
  size CDATA #REQUIRED>
<!ELEMENT brokers (photos)>
<!ATTLIST brokers enabled (true|false) #REQUIRED>
<!ELEMENT watermark (#PCDATA)>
<!ELEMENT picture EMPTY>
<!ATTLIST picture
  size CDATA #REQUIRED
  type (jpeg|gif|png) #REQUIRED
  usewatermark (true|false) #REQUIRED>
<!ELEMENT pictures (watermark?,picture+)>
<!ATTLIST pictures
  enabled (true|false) #REQUIRED
  maxcount CDATA #REQUIRED>
<!ELEMENT econfig (version?,offer?,codebook?,brokers?,pictures?)>
]>
```

## 3. Methods

Methods are used for establishing the communication and for data exchange. Export is initialized on request of the MultiEstate application user. Application subsequently calls methods of the server, which return the result of the given operation and by that it is possible to influence in a certain extent calling of the

other methods. But the biggest part of the form for calling methods is given by configuration of the interface (see in the text above).

All the methods, which contain authentication code in the variable *Authkey* within the input parameters, should ensure verification of this key prior any operation, so the abuse of the interface by an unauthorized person will be prevented.

### 3.1 Data types

**string** – String value, the maximum length of this parameter is stated in the square brackets after this type. If this type is followed by a colon (":"), then the word after the colon identifies table from the CodeBook (see in the text above), which will be used (if it is enabled).

**integer** – Integral absolute value (max.  $2^{32}$ ).

**boolean** – Logical value true/false.

**struct** – Structured value or also associative array, where the keys of this array represent the name of a parameter and the value is then the concrete value of this parameter of a certain data type (can even be struct).

**array** – Similar as struct, but for the keys a numeric array beginning at 0 is used. These numbers don't have any meaning, they are just a collection of similar data.

**base64** – Mostly binary or even other data, with which a lossless transfer is required, coded with Base64 coding.

### 3.2 Structure of the returned value

Each method has to return a structured output (type *struct*), which always contains at least one key with the name *StatusCode*. It contains numeric identification of the result state of the operation. For the standard operation codes from 0 to 500 are reserved and their meaning can be found below in the table "Summary of return codes". Second mutual key in the answer can be *StatusMessage* containing text statement to the result code, which is subsequently displayed to the user of the application. For the standard codes from the table it is not necessary to use this input parameter, but by it it is possible to change the text of the statement.

If you want to report another type of event from your import interface, you can choose a different code outside the reserved range. Then it is convenient to use key *StatusKey* with the explanation of the problem.

Other keys of the returned value vary according to the called method and are stated with each separately in the descriptions below. It is necessary to precisely state their name including letter case and data type.

**Example of the method „check“ result**

```
struct(
  [StatusCode] = 200,
  [ConfigTime] = 0,
  [SendCodeBook] = false
)
```

**Summary of return codes**

Code	Message
200	Operation was successful
300	Authorization code does not comply

### 3.3 Compulsory methods

Here follows a list and descriptions of methods, which have to be implemented on the import server. When any of them is missing a fatal error will occur and the export won't be finished.

**check**

Basic method for verifying availability of the connection with the server and up-to-dated configuration. It is called at the beginning of every export and if it doesn't return a valid result to a certain time-limit, the server is considered unavailable and the export ends with an error.

The input parameter of *InterfaceVersion* is the last version of the export interface. This parameter has an informative nature. If the version you're using and value of this parameter vary, you can request an updated description of the interface from your MultiEstate application provider.

If the output value of *ConfigTime* equals 0, a default configuration will be used for the export (see above) and the method *getConfig* won't be called for acquiring configuration file.

Second return value *SendCodeBook* informs the application to call method *sendCodeBook*, which sends to the server a recent list of all permitted codebooks (for more information see configuration branch codebook).

**INPUT PARAMETERS:**

*InterfaceVersion* *string[10]* – Identification of the export interface version

**OUTPUT VALUES:**

*ConfigTime* *integer* – UNIX timestamp of the time of the last change in configuration XML

SendCodeBook *boolean* – request from the application for the list of currently available codebooks

## getConfig

Method for transfer of configuration XML file to MultiEstate. After receiving the configuration is checked by DTD specification and if something doesn't comply, the method *reportError*, containing description of the problem, is called.

### INPUT PARAMETERS:

AuthKey *string[100]* – Authentication key for the assigned profile

### OUTPUT VALUES:

ConfigData *base64* – Configuration XML file encoded with Base 64

## sendOffer

Sending of the basic information about the offer to the server. This method contains fixed data, which can only change with other versions of the interface and there is no way to configure them (except enabling/disabling CodeBook). Every offer is identified by an unique code within the real-estate agency. If an offer with this code already exists on the server, it's data will be updated, otherwise it will be added. Parameters listed below are compulsory and will always be filled-out.

### INPUT PARAMETERS:

AuthKey *string[100]* – Authentication key for the assigned profile

Code *string[10]* – Unique code of the offer within the real-estate agency

OfferData *struct* – Basic information about the offer in a structured array

Location *struct* – Information about the location of the estate

### OUTPUT VALUES:

OfferId *integer* – Unique numeric Id under which the offer is stored on the server or 0

### struktura parametru offerData

OfferData struct (		
Name	string[1000]	Name of the offer
Description	string[1000]	Description of the offer
Class	string[255]:class	Estate class
Transaction	string[255]:trans	Transaction type (sale, tenancy...)
Stage	string[255]:stage	Stage of the offer
Price	string[20]	Total price of the offer
PriceUnit	string[255]:priceunit	Price unit (m <sup>2</sup> , property...)
Currency	string[5]:currency	Currency (CZK,USD,EUR...)
TotalArea	integer	Total estate area
Contract	string[255]:contract	Type of the contract (exclusive...)
ContractFrom	integer	UNIX timestamp start date of the contract validity
ContractTo	integer	UNIX timestamp end date of the contract validity

FreeDate	string[255]	From when is the offer available
HotOffer	boolean	Attribute indicating interesting offer
Broker	integer	Identifier of the agent (broker)

For the description of the estate a public codebook UIR-ADR is used, it is maintained by the Ministry of labor and social affairs. All the information about this code book can be found at the address <http://forms.mpsv.cz/uir/>. Export interface always sends all the available information about the estate location (except descriptive number). If you don't want to use UIR-ADR codebook, just ignore the codes and use the text form of the various parts of the location.

Structure of the parameter Location		
Location struct (		
RegionId	integer	Region code from the UIR-ADR codebook
RegionName	string[255]	Region name
CityId	integer	City code from the UIR-ADR codebook
CityName	string[255]	Name of the city
DistrictId	integer	District code from the UIR-ADR codebook
DistrictName	string[255]	District name
CityPartId	integer	City part code from the UIR-ADR codebook
CityPartName	string[255]	City part name
StreetId	integer	Street code from the UIR-ADR codebook
StreetName	string[255]	Street name
Cadastral	string[255]	Cadastral area name
)		

## deleteOffer

Method for deleting an offer from the server or for informing the server, that the offer is no longer valid and shouldn't be presented anywhere. If the data are only archived, this method doesn't have to do anything, but it has to be implemented and it has to return a successful execution of the operation.

### INPUT PARAMETERS:

AuthKey *string*[100] – Authentication key for the assigned profile

Code *string*[10] – Unique code of the offer within the real-estate agency

## reportError

Reporting of the problems with the interface. If the problem is detected automatically, this method sends a text description of the problem. In other cases just a general information about an error occurrence. It is in your own interest to gather this errors and record them for later correction. If you are sure, that you've done everything according to the instructions stated in this document and system still reports an error, contact our Technical support

center on the e-mail address [support@multiestate.cz](mailto:support@multiestate.cz) and send us a detailed description of the problem.

**INPUT PARAMETERS:**

*string* ErrorMessage – Text description of the error

### 3.4 Optional methods

Existence of these methods in the server implementation isn't required, because their calling is conditioned by a specific setting of the configuration. Non-existence of the method on the server can only initiate a warning and the export will be even then successfully finished.

#### checkBroker

Method for verification of the agent's existence on the server and for checking, if he's up-to-dated. Calling of this method is conditioned by the authorization of the agent's export according to the configuration. If the agent isn't on the server up-to-dated or he doesn't even exist, the method *sendBroker* is called.

**INPUT PARAMETERS:**

AuthKey *string*[100] – Authentication key for the assigned profile

BrokerId *integer* – Unique Id of the agent (broker) within the application

**OUTPUT VALUES:**

ChangeTime *integer* – UNIX timestamp of the last change of the agent, 0 if the agent doesn't exist

#### sendBroker

Method for adding or updating of an agent on the server. Calling of this method is conditioned, firstly by the permission to send agents and secondly by agent being up-to-dated on the server. Each agent is identified by an unique number within the application. Under the same number also a possible updates of the agent will be sent. This number is also used for connecting offer with an agent (parameter *Broker* in the field *OfferData* in the method *sendOffer*).

**INPUT PARAMETERS:**

AuthKey *string*[100] – Authentication key for the assigned profile

BrokerId *integer* – Unique Id of the agent (broker) within the application

FirstName *string*[255] – First name of the agent

LastName *string*[255] – Surname of the agent

Title *string*[32] – Title of the agent

Phone *string*[32] – Primary phone number of the agent

PhoneOther *string[32]* – Other phone numbers of the agent

Email *string[255]* – E-mail address of the agent

PhotoData *base64* – Photo of the agent in a format according to the configuration

## sendFreeltems

Method for sending additional information about the offer. In the current version all available information is sent.

### INPUT PARAMETERS:

AuthKey *string[100]* – Authentication key for the assigned profile

OfferCode *string[10]* – Unique code of the offer within the real-estate agency

Items *struct* – Sent additional information

### Structure of the input parameter Items

```
Items struct (
  ItemId array (      string[255]:freeitem  Identification of the item in
    value             string[255]:foptions  the key
    ...               List of values
  )
)
```

Identifier of the item is an unique string constant consisting of lower case letters of English alphabet, or numbers. To separate more words a dash is used. Array of values with every item can contain one or more values. If there are more values, the first value (with index 0) is labeled by the application user as a primary, all the others are additional. Each single value is the type *string[255]*.

### Example of the input parameter Items (when using CodeBook)

```
struct (
  estate-condition = array
    good,
    after reconstruction
),
amenities = array(
  description of civic amenities
)
)
```

In this particular example it is appropriate to check, if the item exists in the CodeBook *freeitem*. If it doesn't exist, it is a scalar value and the value on the index 0 in the array under this item is at the same time the final value (usually string or number entered by the user). In other cases, it is necessary to find text formulation for the given option through CodeBook *foptions*.

**Example of the input parameter Items (without the CodeBook)**

```

struct (
  Estate condition = array
    Good,
    After reconstruction
  ),
  Amenities = array(
    Description of civic amenities
  )
)

```

Very similar to previous example, but it is not necessary to find the values in the CodeBook.

**checkPhotos**

Method for reading the list of all exported photos of the given offer. By this method the application checks which photos it should send, update or delete on the server.

**INPUT PARAMETERS:**

AuthKey *string*[100] – Authentication key for the assigned profile

OfferCode *string*[10] – Unique code of the offer within the real-estate agency

**OUTPUT VALUES:**

Photos *struct* – Array with information about photos connected with the offer

**Structure of the output value Photos**

Photos struct (	
PhotoId struct (	One block for every photo
Position integer,	Last recorded position of the photo
LastChange integer	UNIX timestamp of the last change
)	
...	
)	

**sendPhoto**

Method for sending one photo in all requested formats, including connecting it with the offer. Binary data of the photo are sent only at the first import or if there is a change of format in the configuration. During other updates the photo's name and position can be changed.

**INPUT PARAMETERS:**

AuthKey *string*[100] – Authentication key for the assigned profile

OfferCode *string*[10] – Unique code of the offer within the real-estate agency

PhotoId *integer* – Unique numeric identification of the photo

Title *string[255]* – Name of the photo

Position *integer* – Position number of the photo, main photo has value 0

PhotoData *struct* – Indexed field with binary data of the various formats of the photos (key 0 = first format, key 2 = second format, ...) according to the configuration

---

## deletePhoto

---

Method for deleting outdated photos from the offer.

**INPUT PARAMETERS:**

AuthKey *string[100]* – Authentication key for the assigned profile

PhotoId *integer* – Unique numeric identification of the photo

---

## sendCodeBook

---

Method ensuring sending of the up-to-dated CodeBooks. This method is called only if it is requested by the method *check*. Only CodeBooks authorized through the configuration are sent.

**INPUT PARAMETERS:**

AuthKey *string[100]* – Authentication key for the assigned profile

CodeBook *struct* – Structured array with the individual tables of the CodeBook

Array CodeBook contains individual tables. Name of the table is always the key and under this key another structured array is contained, where the individual identifiers are the key and values are their text formulation.