2019

[API Specification]

[API for integrating Voice2Med with HIS]

# Introduction

This specification describes API interaction principles for filling template forms using **Voice2Med** software. The introduction describes access technologies and contains a schematic diagram of API interaction. Other sections feature concrete ways of using **Voice2Med** API to fill forms.

This specification is aimed at hospital information system (**HIS**) developers that need to integrate **HIS** with **Voice2Med** software and can be used as a guide.

##### Please note that document version [v.1.1] is not final. API can be changed.

## Used technologies

**Voice2Med** API uses **WebSocket** full-duplex communication protocol for integration with **HIS**. Using full-duplex communication means that after **Voice2Med** connects to **HIS**, **Voice2Med** (acting as a server) can notify **HIS** about the results of recognition.

As most operations belong to synchronous request model, the option of using REST API for such requests is also provided.

## Requirements and restrictions for direct connection to the server

This specification mainly concerns connection to the **Voice2Med** client side using localhost:33999 and localhost:39256 ports. In this case, sound acquisition, sound transmission to the server and some other operations are performed by **Voice2Med** client. In some cases, it is possible to directly connect **HIS** to the server. This section describes requirements and restrictions for the direct connection.

### Connection ports

If you connect to the server directly, you should use the ports different from the client ports.

|  |  |  |
| --- | --- | --- |
|  | **Client port** | **Server port** |
| **WebSocket** | localhost : 33999 | 192.168.x.x : 34000 |
| **Rest** | localhost : 39256 | 192.168.x.x : 39255 |

### Client identifier

If you connect to the server directly (not using **Voice2Med** client), it is necessary to send client identifier in cookies. The identifier is a GUID sent as a string. The identifier should not be changed during application functioning and should remain the same from launch to launch. The client version and application name should be also sent in cookies.

|  |  |  |
| --- | --- | --- |
| **Key** | **Example value** | **Description** |
| DESKTOP\_ID | a2906915-e21c-4747-971b-8b82b43c0d2a | Generated by the client once during the first software launch |
| VERSION | 0.3 | Temporarily not used. At this moment you need to send **0.3** value. |
| CLIENT\_APP\_NAME | QT\_Voice2Med\_Client | Text with application name |

### Code offsets of messages sent through API

Client-server messages have an offset of 100. For example, to pre-import a catalog, you need to send message with code 101, not code 1 (as in case of client connection).

Server-client messages have an offset of 200. For example, the notification about changing the recognition mode will come with code 200, not code 0 (as in case of client connection).

### Choosing language model

At the start of recognition process, **Themeld** field should be filled (unlike in case of client connection). For more information about filling the field, see [Starting form recognition](#7jho5b3dkobr) subsection.

### Additional messages

If you connect to the server directly, you will get messages not described in this specification. Some messages are mandatory and useful (such as messages about sending the sound or absence of license), some of them are useful but not mandatory (speech synthesis data), some can be ignored.

#### Sending sound from microphone to the server

If you connect to the server directly, the tasks of sound acquisition and sending the sound to the server are performed in your application. The sound should be acquired with the sampling frequency of 16 kHz.

Then the sound is sent to the server by **WebSocket** connection in binary format.

#### User notification about an event (text message)

User can receive messages from the server side. You can ignore these messages or process them and show them to the user.

Example of a message:

###### {

######  "MessageType": 5,

######  "Data": {

######  "Message": "Message to the user",

######  "Type": 3,

######  "Action": 1

######  }

###### }

|  |
| --- |
| **Type: type of a message** |
| Information message | 1 |
| Warning | 2 |
| Error | 3 |

|  |
| --- |
| **Action: additional action** |
| Does not exist | 0 |
| Reconnect or close the application (appears when you do not have application license or in case of another critical error). | 1 |
| Stop recognition process (appears when you do not have appropriate license). | 2 |

#### Getting information about license

When you connect to the server, it returns information about license. Use this message to check the application launch capability or to learn more about your license status.

Example of a message:

###### {

######  "MessageType": 7,

######  "Data": {

###### "ProtectionData": {

######  "LastUpdateTimestamp": "2019-01-09T17:10:28.1875096+03:00",

######  "CountOfCapturedLicenses": 0, // [deprecated]: not used

######  "CountOfLicenses": 1, // [deprecated]: not used

######  "DaysLeft": 1, // Number of days left

######  "LaunchCount": -1, // [deprecated]: not used

######  "LicenseType": 1, // 0 - undefined, 1 - Trial, 2 - permanent

######  "Type": 2 // 0 - undefined

######  },

######  "Type": 0, // [deprecated]: interested only in 0 type messages

######  "ComponentId": "endoscopy\_2.3.95" // theme name or empty string

######  }

###### }

#### managing speech synthesis parameters

Example of a request for the list of available voices:

###### {

######  "MessageType": 21,

######  "Data": {}

###### }

Example of a request for installing additional **TTS** parameters:

###### {

######  "MessageType": 23,

######  "Data": {

######  "SelectedVoice": {

######  "Name": "Alexander"

######  },

######  "Speed": 1.4

######  }

###### }

Example of getting server information about speech synthesis component status (same as [[TTS](#77ftla24ckmr) status message](#77ftla24ckmr)):

###### {

######  "MessageType": 22,

######  "Data": {

######  "VoiceList": [

######  {

######  "Name": "Alexander"

######  },

######  {

######  "Name": "Julia"

######  }

######  ],

######  "SelectedVoice": {

######  "Name": "Alexander"

######  },

######  "Speed": 1.4,

######  "State": 0

######  }

###### }

Sound from **TTS** comes in binary format with the frequency of 22050.

## Schematic diagram of interaction

To enable the form filling mode, **HIS** should connect to **Voice2Med** through **WebSocket** (localhost:33999 by default) and then send the message containing metadata with the form description (see more in [Form initialization](#7jho5b3dkobr) subsection). After successful processing of metadata, the client (**HIS**) will receive [the notification about changing the currently active form](#56hv5asf5xjw). Form initialization does not lead to the change of recognition mode. If the voice activation mode was active from the start, it will not change. If the continuous recognition mode was active, it will automatically change to form filling mode. You will receive a [respective message](#56hv5asf5xjw).

**If metadata processing fails, Voice2Med will send a respective message (object:** [**Message<ErrorInfo>**](#4gsl1mokhp18)**)**



Figure 1: Interaction of Voice2Med with HIS

In this mode, **Voice2Med** uses form grammar to recognize the text pronounced by an operator. Then **Voice2Med** sends the messages to **HIS** describing the fields to be filled and the data for filling the fields. After filling all fields, **HIS** [informs Voice2Med about completing the form filing task](#v4kfjag5qyk1) and [receives notification about changing the recognition mode](#56hv5asf5xjw) in response.

## Voice2Med operation modes

Currently the system supports three modes of speech recognition:

1. Voice activation mode. This is a default mode that allows users to switch to the different mode using voice command.
2. Continuous recognition mode. This mode is intended for filling simple text documents. When it is enabled, the active window in focus receives the recognized text in the form of system key logs.
3. Form filling mode (standard). This mode is intended for filling a concrete form using field metadata that was previously sent to **Voice2Med**. After the field name and value are recognized in user’s speech, the system sends the respective notification to the application through **WebSocket** and 33999 port.

This specification describes standard form filling mode.

1. Form filling mode (**Voice Wizard**). In this mode, user should fill fields in a certain order. The application pronounces the name of a current field and waits for the user to pronounce its value (without the name). This mode speeds up form initialization, increases the accuracy of speech recognition, and ensures sequential form filling.

This specification describes only form filling modes. Please note under which conditions and how **Voice2Med** switches to form filling modes.

### Turning on form recognition mode from voice activation mode

Suppose the application is launched, the language model is chosen, and the microphone is enabled. In this case, the following ways of switching to the form recognition mode are applicable:

* Performing the command for [form initialization/form choice](#7jho5b3dkobr) and then activating recognition mode. First you send the form metadata (or form key, if the form was imported or loaded before that) and then you enable recognition mode by voice (the default command is “Ok, Voco”) or by double-pressing **Ctrl**.
* Performing [starting form recognition command](#_Starting_form_recognition).

### Turning on form recognition mode from continuous recognition mode

Suppose the application is launched and the continuous recognition mode is started by voice. In this case, to switch to the form recognition mode, you should perform [form initialization/form choice](#7jho5b3dkobr) command or [starting form recognition command](#_Starting_form_recognition).

### Exiting form recognition mode

To exit form recognition mode, you need to perform [finishing form editing command](#v4kfjag5qyk1). In this case, continuous recognition mode will be activated.

## Message container

Any message sent between **Voice2Med** and **HIS** through WebSocket is put into **Message<T>,** a typed class container. See **Message<T>** specification below:

|  |
| --- |
| **Message<T>** |
| Class container with respective data |
| **Field** | **Filed type** | **Requirements** | **Description** |
| MessageType | Int32 | Mandatory | [Message type](#4u9p8o9jti02) |
| Data | T | Mandatory | Data matching the message type |

## Catalog of message types

### Client 🡪 Server

|  |  |  |
| --- | --- | --- |
| **Name** | **Message code** | **Returned synchronizing message** |
| [Request for catalog pre-import](#dmoqxr2yil9i) | 1 | Server → client, code: 4orServer → client, code: 5 |
| [Request for form pre-import](#_pre-import_of_form) | 2 | Server → client, code: 4orServer → client, code: 5 |
| [Request for getting the list of cached metadata](#sz242wufzyfu) | 3 | Server → client, code: 3orServer → client, code: 5 |
| [Request for getting information about cached metadata by identifier](#drhnviprpify) | 4 | Server → client, code: 4orServer → client, code: 5 |
| [Request for setting currently active group or element](#46awfdze8jro) | 5 |  |
| [Starting form recognition command](#7jho5b3dkobr) | 6 | Server → client, code: 0orServer → client, code: 5 |
| [Finishing form editing command](#v4kfjag5qyk1) | 7 | Server → client, code: 0orServer → client, code: 5 |
| [Request for getting the list of themes (language models)](#ifrggsxethgr) | 8 | Server → client, code: 7orServer → client, code: 5 |
| [Request for changing themes (language models)](#u2g4cobhpro) | 9 | Server → client, code: 8orServer → client, code: 5 |

### Server 🡪 Client

|  |  |  |
| --- | --- | --- |
| **Name** | **Message code** | **Returned synchronizing message** |
| [Notification about changing current recognition mode](#56hv5asf5xjw) | 0 |  |
| [Notification about value recognition](#8msa0ucpkfko) | 1 |  |
| [Notification about focusing on group or element](#8wj8i9r9hxfo) | 2 |  |
| [Returning the list of cached metadata](#o6433qazhgt0) | 3 |  |
| [Returning the description of cached metadata](#p62b31a988t2) | 4 |  |
| [Error notification](#4gsl1mokhp18) | 5 |  |
| [Notification about recognizing several values (in case the field cannot be clearly determined](#pu6egxb6c16d)) | 6 |  |
| [Returning the list of available language models](#s6h5wd0kxeb) | 7 |  |
| [Notification about changing currently active language model](#cszg9kere2lf) | 8 |  |
| [Notification about preliminary results of processing](#63gzyberhe2y) | 9 |  |
| [Notification about command activation](#8d3rbf7wdg9g) | 10 |  |
| [Notification about TTS status](#77ftla24ckmr) | 11 |  |
| [Notification about activation of enabling recognition command](#t40usxkda95z)  | 12 |  |
| [Notification about activation of disabling recognition command](#t40usxkda95z) | 13 |  |

# Form filling

This section contains information about data types and data communication between **Voice2Med** and **HIS**. First subsection provides the [list of recognized data types](#vrjnsehh7j7k), their short description and the links to respective subsections with examples of field filling. Next subsection describes formats of form metadata and concentrates on form initialization process that comes before form filling.

Another important part of form filling is focus management and element grouping. It is described in the [respective subsection](#x3d2jjnfpuet).

[Next subsections](#8msa0ucpkfko) describe interaction with different field types in more detail.

## Data types available for recognition

|  |  |  |
| --- | --- | --- |
| **Data type** | **Code** | **Description** |
| [String](#8msa0ucpkfko) | 0 | Simple text data |
| [Boolean](#k57uoh8n9kab) | 1 | Boolean value (true/false) |
| [Int32](#6cwglamh1ff2) | 2 | IntegerMinimum value:-2 147 483 648;Maximum value: 2 147 483 647; |
| [Decimal](#fu5c8xhh6p3k) | 3 | Decimal fraction |
| [Date](#ou1xem7wx8re) | 4 | Date |
| [Time](#ou1xem7wx8re) | 5 | Time |
| [DateTime](#ou1xem7wx8re) | 6 | Date and time |
| [Link](#gopgtdgxuqq7) | 7 | Catalog or link to the catalog on client’s side (HIS) |
| [TimeSpan](#5kglnajn9m5z) | 8 | Time period, for example:“One year”“Five years”“Four years and seven months”“Twelve months”“Eight hours” |
| [Complex](#jgfxv4lsw1rl) | 100 | Complex field |

## Starting form recognition

Before starting voice form filling, it is necessary to send form metadata that will be used for recognition. See class diagram describing the structure of form metadata below:



Figure 2: Metadata structure

To send form description (**Message<FormMetadata>** object), you need to connect to **Voice2Med** through WebSocket (localhost: 39999 by default) and send this object in JSON format.

### Example

Example of metadata production for the following form:



Figure 3: Form with medical data

Below you see full example of the JSON message that you need to send to **Voice2Med**. The metadata is described by [**FormMetadata**](#hvb1hwe1jqfl) object, that is put into [**Message<FormMetadata>**](#lolkc6r0g7ah) container.

###### {

######  "MessageType": 6,

######  "Data": {

######  "Key": "a0bc1744-010c-4414-84e6-acabeff7b517",

###### "Name": "Form №1",

######  "Childs": [

######  {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  },

######  {

######  "Key": "1",

######  "Value": "Female"

######  }

######  ],

######  "Type": 7,

######  "Synonyms": [

######  "Sex"

######  ],

######  "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

######  },

######  {

######  "Type": 1,

######  "Synonyms": [

######  "Insurance status",

######  "Insurance"

######  ],

######  "Key": "1d77096e-53eb-48ea-bfad-4ea3bab2b3a7"

######  },

######  {

######  "Childs": [

######  {

######  "Type": 2,

######  "Synonyms": [

######  "Pulse"

######  ],

######  "Key": "b5dcd5b9-576a-420b-aea2-b81171a2ee8e"

######  },

######  {

######  "Childs": [

######  {

######  "HandbookKey": "3b00e940-1f75-4bae-ae33-7796dbceb768",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Normal"

######  },

######  {

######  "Key": "1",

######  "Value": "High"

######  },

######  {

######  "Key": "2",

######  "Value": "Low"

######  }

######  ],

######  "Type": 7,

######  "Synonyms": [],

######  "Key": "ec58154c-28d7-4aaa-953e-7a7fc149ee56"

######  },

######  {

######  "Type": 2,

######  "Key": "7c11c54e-4cef-44f5-b895-2c82383d151d"

######  },

######  {

######  "Type": 2,

######  "Synonyms": [

######  "by"

######  ],

######  "Key": "92e99d18-54e5-4920-89d3-70d589f312e2"

######  }

######  ],

######  "Type": 100,

######  "Synonyms": [

######  "Blood pressure"

######  ],

######  "Key": "06b13fe2-32e6-49a1-b376-ff0237414a6b"

######  }

######  ],

######  "Synonyms": [

######  "Before activity",

######  "Before"

######  ],

######  "Key": "34041bd0-8fac-4026-ac9f-3ede5dc23f50"

######  },

######  {

######  "Childs": [

######  {

######  "Type": 2,

######  "Synonyms": [

######  "Pulse"

######  ],

######  "Key": "8974d016-0669-4b1f-a968-ad124240c2dd"

######  },

######  {

######  "Childs": [

######  {

######  "HandbookKey": "8ecafe5b-a6cf-4576-b423-cd915d25bba3",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Normal"

######  },

######  {

######  "Key": "1",

######  "Value": "High"

######  },

######  {

######  "Key": "2",

######  "Value": "Low"

######  }

######  ],

######  "Type": 7,

######  "Synonyms": [],

######  "Key": "aa2d7231-4176-45cf-bb02-b556a16223b3"

######  },

######  {

######  "Type": 2,

######  "Key": "34110649-7e2d-4f0e-9d19-5c0734c47509"

######  },

######  {

######  "Type": 2,

######  "Synonyms": [

######  "by"

######  ],

######  "Key": "c6e21691-6116-4097-8b87-824f376508a2"

######  }

######  ],

######  "Type": 100,

######  "Synonyms": [

######  "Blood pressure"

######  ],

######  "Key": "af298d98-3f3c-4529-8ed1-0bc2857c6bd8"

######  }

######  ],

######  "Synonyms": [

######  "After activity",

######  "After"

######  ],

######  "Key": "ea0ac1b8-d91d-4b98-8d63-4f565048ab77"

######  },

######  {

######  "Type": 0,

######  "Synonyms": [

######  "Description"

######  ],

######  "Key": "9cb3cf80-8f5d-4bdd-9956-16aecb405f3a"

######  },

######  {

######  "DateTimeFormat": "dd.MM.yyyy",

######  "Type": 4,

######  "Synonyms": [

######  "Scheduled date of next checkup",

######  "Next checkup",

######  "Date of checkup",

######  "Date of next checkup"

######  ],

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }

######  ],

######  "CultureCode": "ru-RU"

######  }

###### }

After successful loading of form metadata, the client will receive a [message about changing current recognition mode](#56hv5asf5xjw). If it is impossible to initialize the form (for example, if metadata was produced incorrectly), [error message](#4gsl1mokhp18) will be sent.

### Example of a request for starting pre-imported form recognition

###### {

######  "MessageType": 6,

######  "Data": {

######  "Key": "a0bc1744-010c-4414-84e6-acabeff7b517",

######  "CultureCode": "ru-RU"

######  }

###### }

##### If the form and all its catalogs were pre-initialized, then during [form initialization](#7jho5b3dkobr) you can exclude Childs field from the data you send in [FormMetadata](#hvb1hwe1jqfl) object.

Read more about pre-import of forms and catalogs in the [respective section](#xdhi4zyygo1x).

### FormSettings: additional parameters of form recognition

**FormSettings** is an optional field that allows to influence additional parameters of form recognition, such as **TTS** management (voice confirmation of recognition) and changing focus on elements.

Example of data:

###### {

######  ...

######  "FormSettings": {

######  "TtsFieldNameEnabled": true,

######  "TtsFieldValueEnabled": false,

######  "TtsTextFieldValueEnabled": true,

######  "CyclicFocusNotificationEnabled": true,

######  "WizardMode\_TtsFullGroupNamingEnabled": false,

######  "WizardMode\_IsAutoSwitchingEnabled": true,

######  "WizardMode\_IsAutoSwitchingEnabledForText": true

######  },

######  ...

###### }

|  |
| --- |
| **FormSettings** |
| Class that contains additional form recognition parameters |
| **Field** | **Field type** | **Default value** | **Description** |
| TtsFieldNameEnabled | bool | true(starting from version 4.1.216: false) | Enable/disable voicing of field name |
| TtsFieldValueEnabled | bool | false | Enable/disable voicing of field value |
| TtsTextFieldValueEnabled | bool | true(starting from version 4.1.215: false) | Enable/disable voicing of field value for text fields.*Only if* ***TtsFieldValueEnabled*** *=* ***true.*** |
| WizardMode\_TtsFullGroupNamingEnabled | bool | false | *Used only for* ***VoiceWizard*** *mode.***True:** full voicing of all the groups with the field when you move to the field.**False:** group name will be voiced only if previous active element belonged to a different group. |
| WizardMode\_IsAutoSwitchingEnabled | bool | true | *Used only for* ***VoiceWizard*** *mode.***True:** after filing the field with value (successful recognition) the system automatically switches to the next field.**False:** the system does not automatically switch to the next field. |
| WizardMode\_IsAutoSwitchingEnabledForText | bool | true | *Used only for* ***VoiceWizard*** *mode*. *Only if* ***WizardMode\_IsAutoSwitchingEnabled*** *=* ***true****.*Enable/Disable automatic switch for text fields. |
| CyclicFocusNotificationEnabled | bool | true | Enable/Disable cyclical notification about changing focus on element.Determines whether the client needs to be informed about change of focus on element (**ActiveElement**), if the client initiated the change. |

### Mode: form filling mode

**Mode** is an optional field that allows to choose recognition mode: **VoiceWizard** or default mode.

Example of data:

###### {

######  ...

######  "Mode": 1,

######  ...

###### }

It is an additional field in **FormMetadata** object.

### Custom commands

**Custom commands** is an optional field that allows you to set your own voice commands. It is a collection of phrases and command identifiers. When the phrase that matches the command is recognized, the client receives the [respective notification](#8d3rbf7wdg9g) with the command identifier.

Example of data:

###### {

######  ...

######  "CustomCommands": [

######  {

######  "Key": "7d4565a7-b408-44d4-afc4-26b96a842446",

######  "Value": "Clear field"

######  },

######  {

######  "Key": "037ccb5c-6bde-4853-b7df-a6bb5b72916e",

######  "Value": "Next field"

######  },

######  {

######  "Key": "d7823973-af46-4cdc-bab8-1c8db097ad7a",

######  "Value": "Previous field"

######  },

######  {

######  "Key": "546f4ef9-ff59-4a5b-a368-a97c3e03240c",

######  "Value": "Finish form recognition"

######  }

######  ],

######  ...

###### }

######

It is an additional filed in **FormMetadata** object.

### Theme (language model)

**Themeld** is an optional field that allows to specify language model to be used in process of form recognition.

##### Language model (or theme) influences only filling fields with direct cursor, such as string fields and combo boxes that allow direct cursor (not a default setting). The influence depends on the value of direct cursor.

##### For example, if we have Comment field:

##### 1) The user says “Comment” and the focus is set on Comment field (language model does not have influence).

##### 2) The user says: “This field will contain some random text” (language model has influence and the result of recognition depends on it).

##### Language model does not influence filling other field types.

Example of data:

###### {

######  ...

######  "ThemeId": "main\_2.3.13",

######  ...

###### }

######

It is an additional field in **FormMetadata** object.

You can read about getting the list of available themes [in the respective subsection](#ifrggsxethgr). Please note that there are two ways of getting the themes: through **WebSocket** and REST API.

### Description of classes, class fields and special requirements

|  |
| --- |
| **FormMetadata** |
| Class that contains full form description |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Form type identifier. It is used for [caching metadata](#xdhi4zyygo1x). |
| Name | String |  | Name |
| Childs | List of FormElementMetadata | Mandatory | Collection (hierarchy) of form elements |
| CultureCode | String |  | [Culture code](https://msdn.microsoft.com/ru-ru/library/ee825488%28v%3Dcs.20%29.aspx) that will be used for serializing the recognized data, such as “ru-RU”. If you do not set culture code, [InvariantCulture](https://msdn.microsoft.com/ru-ru/library/system.globalization.cultureinfo.invariantculture%28v%3Dvs.110%29.aspx) will be used. |
| FormSettings | FormSettings |  | [Additional recognition parameters](#5pwp8yvft0wo)  |
| Mode | Int32 (nullable) |  | 0 – voicing in key-value format (used by default);1 – **VoiceWizard** mode. System pronounces the key and expects the value to be pronounced by the user. |
| CustomCommands | List<KeyValuePair<string, string>> |  | [User commands for managing the form](#2mlbgmklokzv) |
| ThemeId | String |  | [Theme identifier](#ucowkdxazmws) |

|  |
| --- |
| **FormElementMetadata** |
| **Abstract class****Concrete implementations:** [**ContainerMetadata**](#pgfp843rl9he) **and** [**ValuedFormElementMetadata**](#148oagti0tgl)Describing the element in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Form type identifier.This identifier will be used for [caching metadata](#xdhi4zyygo1x). |
| Synonyms | List of strings | Mandatory | Collection of strings used by the system to determine whether the recognized data matches the element. |
| Postfix | List of strings | Not mandatory | Collection of strings (phrases) that can be pronounced after the value and that user usually pronounces, such as units of measurement. For example, in the phrase “Temperature is 37 degrees” the word “degrees” is not necessary for pronunciation and does not influence the final field value (37), but the user is likely to pronounce it.The field is not used for **string** type and containers. |
| **ContainerMetadata** |
| Class container for other elements |
| Childs | List of FormElementMetadata | Mandatory | Collection (hierarchy) of elements in form |
| **ValuedFormElementMetadata** |
| Class with the description of the filled field. **ContainerMetadata** child |
| Type | Int32 | Mandatory | [Field type](#vrjnsehh7j7k) |
| The following fields are relevant only for [Date](#ou1xem7wx8re), [Time](#ou1xem7wx8re) and [DateTime](#ou1xem7wx8re) types |
| DateTimeFormat | String | Used for:[Date](#ou1xem7wx8re)[Time](#ou1xem7wx8re) [DateTime](#ou1xem7wx8re) | If the data type is a data or/and time, the field should contain the [format](https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/custom-date-and-time-format-strings)  of returned data.If you do not set the field, the [InvariantCulture](https://msdn.microsoft.com/ru-ru/library/system.globalization.cultureinfo.invariantculture%28v%3Dvs.110%29.aspx) will be used. |
| The following fields are relevant only for [Link](#gopgtdgxuqq7) type |
| HandbookKey | String | Used for:[Link](#gopgtdgxuqq7) | If the data type is a link or a listing, the field should contain catalog key that identifies the catalog. If the key is not sent, the metadata will not be cached. |
| AvailableValues | List of elements consisting of KeyValuePair<string, string> | Mandatory for:[Link](#gopgtdgxuqq7) | If the data type is a link or a listing, the field should contain the list of possible values. |
| The following fields are relevant for [Complex](#jgfxv4lsw1rl) type |
| Childs | The list of ValuedFormElementMetadata | Mandatory for:[Complex](#jgfxv4lsw1rl) | If the data type is complex, the field should contain the list of simple elements (keys are not necessary). This field type is filled with phrases pronounced continuously, such as “Elevated blood pressure (140 over 90)”.  |

## Managing focus and grouping elements

Elements in the form can be grouped differently. Two different groups can contain the same elements (see [example](#_Example) where **After activity** and **Before activity** groups contain the same field **Pulse**). Bookmarks and other elements can also be viewed as groups.

To make filling these forms easier, the active group concept is used. At the start, the active group is empty. When the operator pronounces the phrase “Before activity”, the respective group is marked as active. If the operator continues and pronounces the phrase “Pulse is 68”, the client (**HIS**) will receive the notification about filling the respective field in the group. The field in the next group (**After activity**) will be ignored.

##### To make another group active, you should pronounce its name (in this case, “Before activity”)

##### To make the group inactive, you should move to filling the field that does not belong to the group (for example Sex or Description)

### Notification about focusing on group or element

Notification comes from a server (**Voice2Med**) to the client (**HIS**). Notification comes when the operator pronounces the name of a group and the group becomes active (see the [example](#_Example) and [**ContainerMetadata**](#pgfp843rl9he) description).

###### {

######  "MessageType": 2,

######  "Data": {

######  "Key": "ea0ac1b8-d91d-4b98-8d63-4f565048ab77"

######  }

###### }

### Returned notification about focusing on group or element

The notification comes from client (**HIS**) to server (**Voice2Med**). It is used when you need to manually set active group (for example, when user switched the tab in form). Any element can be made active.

###  Example

###### {

######  "MessageType": 5,

######  "Data": "ea0ac1b8-d91d-4b98-8d63-4f565048ab77"

###### }

##### If you need to reset the status of active group to inactive, send null in Key field.

###### {

######  "MessageType": 5,

######  "Data": null

###### }

## Filling text forms: notification about value recognition

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of a text field and its value in operator’s speech. The value contains simple text.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Type": 0,

######  "Synonyms": [

######  "Description"

######  ],

######  "Key": "9cb3cf80-8f5d-4bdd-9956-16aecb405f3a"

###### }

###### ...

### Example of a returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 0,

######  "IsFinal": true,

######  "Value": "Some text description",

######  "Key": "9cb3cf80-8f5d-4bdd-9956-16aecb405f3a"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<String>** |
| Class that contains information about text field in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of the element in the form |
| Type | Int32 | Mandatory | [Always equals 0](#vrjnsehh7j7k) |
| Value | String | Mandatory | Recognized value |
| IsFinal | Boolean |  | **true** – final result.**false** – result can be corrected (for example, this result can be shown to user in a tooltip to inform that recognition is in process). |

## Filling Boolean fields (Boolean)

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of a Boolean field and its value. It contains **true** or **false** values.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Type": 1,

######  "Synonyms": [

######  "Insurance status",

######  "Insurance"

######  ],

######  "Key": "1d77096e-53eb-48ea-bfad-4ea3bab2b3a7"

###### }

###### ...

### Example of a returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 1,

######  "Value": true,

######  "Key": "1d77096e-53eb-48ea-bfad-4ea3bab2b3a7"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<Boolean>** |
| Class that contains information about Boolean field in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of the element in the form |
| Type | Int32 | Mandatory | [Always equals 1](#vrjnsehh7j7k)  |
| Value | Boolean | Mandatory | Recognized value |

## Filling integer fields (int32)

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of an integer field and its value.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Type": 2,

######  "Synonyms": [

######  "Pulse"

######  ],

######  "Key": "b5dcd5b9-576a-420b-aea2-b81171a2ee8e"

###### }

###### ...

### Example of a returned value for int32

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 2,

######  "Value": 500,

######  "Key": "b5dcd5b9-576a-420b-aea2-b81171a2ee8e"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<Int32>** |
| Class that contains information about integer field in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of the element in the form |
| Type | Int32 | Mandatory | [Always equals](#vrjnsehh7j7k) 2 |
| Value | Int32 | Mandatory | Recognized value |

## Filling decimal fields (decimal)

Notification about successful recognition comes from the server (**Voice2Med**) to client (**HIS**) when the system recognizes the name of a decimal field and its value in operator’s speech.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Type": 3,

######  "Synonyms": [

######  "Temperature"

######  ],

######  "Key": "93d60cc9-fa05-4cdc-b9dd-628e95fe51d9"

###### }

###### ...

### Example of a returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 3,

######  "Value": 35.42,

######  "Key": "08a39d7e-4620-415d-bbb6-460794e4fcd6"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<Decimal>** |
| Class that contains information about decimal field in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of the element in the form |
| Type | Int32 | Mandatory | [Always equals 3](#vrjnsehh7j7k) |
| Value | Decimal | Mandatory | Recognized value |

## Filling date and time fields (DateTime)

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of a date and/or time field and its value in operator’s speech. The value is returned in string format (specified in metadata).

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "DateTimeFormat": "dd.MM.yyyy",

######  "Type": 4,

######  "Synonyms": [

######  "Scheduled date of next checkup",

######  "Next checkup",

######  "Date of checkup",

######  "Date of next checkup"

######  ],

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

###### }

###### ...

### Example of a returned value for date

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 4,

######  "Value": "20.12.2018",

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }

###### }

### Example of a returned value for time

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 5,

######  "Value": "14:45",

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }

###### }

### Example of a returned value for DateTime

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 6,

######  "Value": "20.12.2018 14:45",

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<String>** |
| Class that contains information about field in the form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of the element in the form |
| Type | Int32 | Mandatory | [Possible values: 4, 5, 6](#vrjnsehh7j7k)  |
| Value | String | Mandatory | Recognized value |

## Filling enumeration value fields (Enum or link to catalog)

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of an enumeration field (or a field with the link to catalog) and its value in operator’s speech. The returned value is the list of **KeyValuePair<string, string>** values.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  },

######  {

######  "Key": "1",

######  "Value": "Female"

######  }

######  ],

######  "Type": 7,

######  "Synonyms": [

######  "Sex"

######  ],

######  "FreeTextAvaliable": false,

######

######  "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

###### }

###### ...

##### Please pay attention to FreeTextAvailablefield. It is false by default, but you can set true value in exceptional cases. Then you can enter free text in the field, not only choose between default options.

##### Free text mode requires special text entry method: first, the user should pronounce field name, then pause and make sure [the field is active](#8wj8i9r9hxfo). Then the user pronounces random text.

### Example of a standard returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 7,

######  "HandbookRow": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  }

######  ],

###### "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

######  }

###### }

### Example of a free returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 7,

######  "Value": "Some random text",

###### "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<KeyValuePair<string, string>[]>** |
| Class that contains information about enumeration field in form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of element in form |
| Type | Int32 | Mandatory | [Always equals 7](#vrjnsehh7j7k) |
| HandbookRow | KeyValuePair<string,string> | Not mandatory (one of the HandbookRow and Value fields will be filled) | Recognized value |
| Value | String | Not mandatory | *If this combo box was chosen as a currently active element (its name was pronounced with a pause) and only the text was recognized (without key-binding or field-binding), the field gets back the value without the key.**It can be used for* ***ComboBox*** *elements with free entry fields.* |

## Filling age type fields (TimeSpan)

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of a time span field and its value in operator’s speech. The value is returned in string format.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by[**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Type": 8,

######  "Synonyms": [

######  "Age"

######  ],

######  "Key": "5d3c09ca-78ee-486d-b278-9c575177c1d7"

###### }

###### ...

### Example of a returned value for TimeSpan

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 8,

######  "Value": "35 years old",

######  "Key": "5d3c09ca-78ee-486d-b278-9c575177c1d7"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<String>** |
| Class that contains information about field in form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of element in form |
| Type | Int32 | Mandatory | [Possible values: 8](#vrjnsehh7j7k)  |
| Value | String | Mandatory | Recognized value |

## Filling complex fields

Notification about successful recognition comes from the server (**Voice2Med**) to the client (**HIS**) when the system recognizes the name of a complex field and its value in operator’s speech. The returned value is the list of **ValuedElementInfo<T>** values.

### Metadata format

During [form initialization](#7jho5b3dkobr) this type of fields is described by [**ValuedFormElementMetadata**](#148oagti0tgl) class.

###### ...

###### {

######  "Childs": [

######  {

######  "HandbookKey": "3b00e940-1f75-4bae-ae33-7796dbceb768",

######  "AvailableValues": [

######  { "Key": "0", "Value": "Normal" },

######  { "Key": "1", "Value": "High" },

######  { "Key": "2", "Value": "Low" }

######  ],

######  "Type": 7,

######  "Synonyms": [ ],

######  "Key": "ec58154c-28d7-4aaa-953e-7a7fc149ee56"

######  },

######  {

######  "Type": 2,

######  "Key": "7c11c54e-4cef-44f5-b895-2c82383d151d"

######  },

######  {

######  "Type": 2,

######  "Synonyms": [

######  "by"

######  ],

######  "Key": "92e99d18-54e5-4920-89d3-70d589f312e2"

######  }

######  ],

######  "Type": 100,

######  "Synonyms": [

######  "Blood pressure"

######  ],

######  "Key": "06b13fe2-32e6-49a1-b376-ff0237414a6b"

###### }

###### ...

### Example of a returned value

###### {

######  "MessageType": 1,

######  "Data": {

######  "Type": 100,

######  "Childs": [

######  {

######  "Type": 7,

######  "Value": [

######  {

######  "Key": "0",

######  "Value": "Normal"

######  }

######  ]

######  },

######  { "Type": 2, "Value": 120 },

######  { "Type": 2, "Value": 80 }

######  ],

######  "Key": "06b13fe2-32e6-49a1-b376-ff0237414a6b"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ValuedElementInfo<ValuedElementInfo<T>[]>** |
| Class that contains information about complex field in form |
| **Field** | **Field type** | **Requirements** | **Description** |
| Key | String | Mandatory | Key identifier of element in form |
| Type | Int32 | Mandatory | [Always equals 100](#vrjnsehh7j7k)  |
| Childs | List of ValuedElementInfo<T> | Mandatory | Recognized value |

## Finishing form editing

After finishing form editing, you need to switch from recognition mode to standard mode.

### Example of a message

###### {

######  "MessageType": 7

###### }

To find example of a message returned to the client, see [the respective section](#56hv5asf5xjw).

## Algorithm of search for the key in form tree

When you fill complex forms by voice, filling fields nested in groups (such as tabs in a form) can be problematic. Filling fields that have the same name is even more problematic. Such fields should be located in different groups, otherwise it will be unclear which field is referred to when the field name is pronounced. To avoid these problems, the system uses *groups of elements in a form* and *focus on the element*. It allows to determine which group elements are filled at the moment.

To find more about managing focus and grouping elements, see the [respective subsection](#x3d2jjnfpuet).

All elements in the form that are described in metadata, can be presented as a tree. The root node is the form itself, simple nodes are the groups and the leaves are the fields.

##### *Tree nodes*: elements that contain child elements.

##### *Leaves*: elements that do not contain child elements.

To find elements that match the user’s phrase in form metadata, a tree walk algorithm is used. The algorithm can be described by the following rule: “When you fill the fields in a group, you should pronounce the group name. Then the group will become active”.

After the user’s phrase is recognized, field filling can follow several hypotheses (if the pronounced phrase matches several fields).

##### *Hypothesis*: a list of groups, fields and their values that match the pronounced phrase.

To determine which field to fill, the system searches through metadata tree. A tree walk starts from the active element (first the form itself). Then by the active element the system finds the parent group. This group can be an element itself if it is not a leaf. After that, for each hypothesis the system chooses the first element of the hypothesis (node) and searches for a matching node or a leaf. If the matching n0de or leaf was found for some element of the hypothesis, all hypotheses without matches are discarded. If nothing was found for any hypothesis in the current node, the system moves to the parent node and repeats the search.

##### Complex field is a node (it has child elements), but it is considered a leaf in process of search for the parent node.

The process continues until all elements of not discarded hypotheses have a match or until the root node is reached (in this case, all hypotheses are discarded). If in the end of the search the only option for filling fields is left, these fields are filled accordingly. If there are several hypotheses left, you get a [respective notification](#pu6egxb6c16d). If there are no hypotheses left, the user’s phrase is ignored.

# Importing catalogs and forms

If you have large amount of metadata to process, production of data can take a lot of time. Form descriptions and catalogs can contain large amount of data; thus, you need to pre-initialize forms and catalogs to speed up production. Then only metadata key identifier will be sent [at the start of form filling](#7jho5b3dkobr), not all data.

### Example of a request for starting pre-imported form recognition

###### {

######  "MessageType": 6,

######  "Data": {

######  "Key": "a0bc1744-010c-4414-84e6-acabeff7b517",

######  "CultureCode": "ru-RU"

######  }

###### }

##### If the form and all its catalogs were pre-initialized, then during [form initialization](#7jho5b3dkobr) you can exclude Childs field from the data you send in [FormMetadata](#hvb1hwe1jqfl) object.

## Pre-importing catalog metadata

Most forms contain simple fields that are filled with recognized values and ref fields that are linked to respective catalogs. To fill this fields by voice, you should send all the values from the linked catalog in key-value format. The same applies to enumeration fields ([for example](#usuubd5gv305), **Sex** field).

Below you can see the example of a request for catalog pre-initialization and the description of sent data fields. To find the example of a received response, see the [special subsection](#p62b31a988t2) of this specification.

### Example of a request

###### {

######  "MessageType": 1,

######  "Data": {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "Name": "Sex",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  },

######  {

######  "Key": "1",

######  "Value": "Female"

######  }]

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **HandbookMetadata** |
| Class that contains catalog metadata |
| **Field** | **Field type** | **Requirements** | **Description** |
| HandbookKey | String | Mandatory | Catalog key identifier |
| Name | String |  | Name |
| AvailableValues | List of elements consisting of KeyValuePair<string, string> | Mandatory | List of possible values in key-value format |

## Pre-importing form metadata

Form metadata can be cached. If the form contains [ref or enumeration](#gopgtdgxuqq7) fields, it is recommended to cache them first and to not send their data while caching. You should send only catalog key and field key.

It means that instead of the following message:

###### ...

###### {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "Name": "Sex",

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  },

######  {

######  "Key": "1",

######  "Value": "Female"

######  }

######  ],

######  "Type": 7,

######  "Synonyms": [

######  "Sex"

######  ],

######  "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

###### }

###### ...

you should send the short version:

###### ...

###### {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "Type": 7,

######  "Synonyms": [

######  "Sex"

######  ],

######  "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

###### }

###### ...

The format of a request for pre-initialization is similar to the format of request for standard [form initialization](#7jho5b3dkobr), except for a different code sent in the [message container](#lolkc6r0g7ah).

Another difference of this method is the format of a returned value. In this case, the recognition mode will not change, but you will receive a message with the [description of cached metadata](#p62b31a988t2).

### Example of request for form pre-initialization

##### In this example all matching catalogs for ref fields were pre-imported.

###### {

######  "MessageType": 2,

######  "Data": {

######  "Key": "a0bc1744-010c-4414-84e6-acabeff7b517",

######  "Name": "Form №1",

######  "Childs": [

######  {

######  "HandbookKey": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "Type": 7,

######  "Key": "7b43b0e0-5e02-4422-984a-e4f531021744"

######  },

######  {

######  "Type": 1,

######  "Synonyms": [

######  "Insurance status",

######  "Insurance"

######  ],

######  "Key": "1d77096e-53eb-48ea-bfad-4ea3bab2b3a7"

######  },

######  {

######  "Childs": [

######  {

######  "Type": 2,

######  "Synonyms": [

######  "Pulse"

######  ],

######  "Key": "b5dcd5b9-576a-420b-aea2-b81171a2ee8e"

######  },

######  {

######  "Childs": [

######  {

######  "HandbookKey": "3b00e940-1f75-4bae-ae33-7796dbceb768",

######  "Type": 7,

######  "Key": "ec58154c-28d7-4aaa-953e-7a7fc149ee56"

######  },

######  {

######  "Type": 2,

######  "Key": "7c11c54e-4cef-44f5-b895-2c82383d151d"

######  },

######  {

######  "Type": 2,

######  "Synonyms": [

######  "by"

######  ],

######  "Key": "92e99d18-54e5-4920-89d3-70d589f312e2"

######  }

######  ],

######  "Type": 100,

######  "Synonyms": [

######  "Blood pressure"

######  ],

######  "Key": "06b13fe2-32e6-49a1-b376-ff0237414a6b"

######  }

######  ],

######  "Synonyms": [

######  "Before activity",

######  "Before"

######  ],

######  "Key": "34041bd0-8fac-4026-ac9f-3ede5dc23f50"

######  },

######  {

######  "Childs": [

######  {

######  "Type": 2,

######  "Synonyms": [

######  "Pulse"

######  ],

######  "Key": "8974d016-0669-4b1f-a968-ad124240c2dd"

######  },

######  {

######  "Childs": [

######  {

######  "HandbookKey": "8ecafe5b-a6cf-4576-b423-cd915d25bba3",

######  "Type": 7,

######  "Key": "aa2d7231-4176-45cf-bb02-b556a16223b3"

######  },

######  {

######  "Type": 2,

######  "Key": "34110649-7e2d-4f0e-9d19-5c0734c47509"

######  },

######  {

######  "Type": 2,

######  "Synonyms": [

######  "by"

######  ],

######  "Key": "c6e21691-6116-4097-8b87-824f376508a2"

######  }

######  ],

######  "Type": 100,

######  "Synonyms": [

######  "Blood pressure"

######  ],

######  "Key": "af298d98-3f3c-4529-8ed1-0bc2857c6bd8"

######  }

######  ],

######  "Synonyms": [

######  "After activity",

######  "After"

######  ],

######  "Key": "ea0ac1b8-d91d-4b98-8d63-4f565048ab77"

######  },

######  {

######  "Type": 0,

######  "Synonyms": [

######  "Description"

######  ],

######  "Key": "9cb3cf80-8f5d-4bdd-9956-16aecb405f3a"

######  },

######  {

######  "DateTimeFormat": "dd.MM.yyyy",

######  "Type": 4,

######  "Synonyms": [

######  "Scheduled date of next checkup",

######  "Next checkup",

######  "Date of checkup",

######  "Date of next checkup"

######  ],

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }

######  ],

######  "CultureCode": "ru-RU"

######  }

###### }

## Request for getting the list of cached metadata

The request is sent from the client (**HIS**) to the server (**Voice2Med**). As a result, you [get data structure](#o6433qazhgt0) that contains the list of pre-imported and cached metadata with catalog key identifiers and HashCode.

##### *HashCode* is a metadata hash calculated on server side. HashCode is intended for metadata version support. HashCode is calculated using metadata JSON and SHA256 algorithm. In case of a form, calculations do not include Themeld field and catalogs’ data (if the catalogs are external, meaning they contain HandbookKey).

### Example of a request

###### {

######  "MessageType": 3

###### }

## Request for getting the description of cached metadata by identifier

The request is sent from the client (**HIS**) to the server (**Voice2Med**). As a result, you [get data structure](#p62b31a988t2)  that contains the description of all pre-imported and cached metadata matching the key identifier in the request.

### Example of a request

###### {

######  "MessageType": 4,

######  "Data": "a0bc1744-010c-4414-84e6-acabeff7b517"

###### }

## Returning the list of cached metadata

The message is sent from the client (**HIS**) to the server (**Voice2Med**). It contains the list of all pre-imported and cached metadata with catalog key identifiers and HashCode.

### Example of a message

###### {

######  "MessageType": 3,

######  "Data": [

######  {

###### "Key": "29df53b0-f9f0-4176-aace-3331fde486a4",

###### "Type": 0,

###### "HashCode": "o�0�[!��\u0014Ճ��[�C=�v-jġ��~F���\u0016"

######  }]

###### }

## Returning the description of cached metadata

The message is sent form the client (**HIS**) to the server (**Voice2Med**). It contains the description of pre-imported and cached metadata matching the key identifier in the request.

### Example of a message

###### {

######  "MessageType": 4,

######  "Data": {

###### "Key": "8b2efc13-5fda-4973-9cd6-2d3e7947addf",

###### "Type": 0,

###### "HashCode": "o�0�[!��\u0014Ճ��[�C=�v-jġ��~F���\u0016"

######  }

###### }

## Request for getting the list of themes

As a result, you get the list of available themes (language models). Language models are used for continuous recognition mode, for example, for filling direct cursor fields. When filling standard fields (without direct cursor), the choice of language model is not important.

### Request of a message to WebSocket

###### {

######  "MessageType": 8

###### }

To find the example of a value returned to the client, see the [respective subsection](#s6h5wd0kxeb).

##### Executing the request through WebSocket can be inconvenient as this request has synchronous logic and WebSocket implies asynchronous communication. Thus, you can [get the list of themes through a synchronous request to REST API](#23hbfnihnhm3). However, it is not mandatory.

## Request for changing theme

As a result, you change currently active language model. Language models are used for continuous recognition mode, for example, for filling direct cursor fields. When filling standard fields (without direct cursor), the choice of language model is not important.

### Example of a message

###### {

######  "MessageType": 9,

######  "Data":

######  {

######  "Id": "Notepad\_respeaking\_16000"

######  }

###### }

# Additional notifications

## Error notifications

Error notifications come from the server (**Voice2Med**) to the client (**HIS**) at the time when the error happens, for example, when you try to load incorrect data as form metadata.

### Example of a returned request

###### {

######  "MessageType": 5,

######  "Data": {

######  "Message": "Incorrect format of loaded metadata"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **ErrorInfo** |
| Class that contains the description of an error |
| **Field** | **Field type** | **Requirements** | **Description** |
| Message | String | Mandatory | Error notification |

## Notification about recognizing several values

The notification comes from the server (**Voice2Med**) to the client (**HIS**) if the recognized value matches several values.

### Example of a returned value

###### {

######  "MessageType": 6,

######  "Data": [

######  [{

######  "Type": 4,

######  "Value": "20.12.2018",

######  "Key": "0d2f0d70-be6a-4f02-b71f-b3e2adc80542"

######  }],

######  [{

######  "Type": 4,

######  "Value": "20.12.2018",

######  "Key": "9707c7c9-5cc3-441b-8418-a840b8bfb356"

######  }]]

###### }

## Returning the list of available language models

Notification contains the list of available language models.

### Example of a returned value

###### {

######  "MessageType": 7,

######  "Data": [

######  {

######  "Id": "Notepad\_respeaking\_16000",

######  "Name": "Notepad\_respeaking\_16000",

######  "Version": "1.0.0"

######  }

######  ]

###### }

## Notification about changing currently active voice model

Notification contains currently active language model. The notification is sent during the change of a language model.

### Example of a returned request

###### {

######  "MessageType": 8,

######  "Data":

###### {

######  "Id": "Notepad\_respeaking\_16000",

######  "Name": "Notepad\_respeaking\_16000",

######  "Version": "1.0.0"

######  }

###### }

## Notification about preliminary results of processing

The notification contains a text with the preliminary results of processing. The text is not the final value that can be used for field filling. It is intended for showing the dynamics of recognition process. If the phrase is long, the user can see the intermediary result and know that the process of speech recognition is active.

### Example of a returned value

###### {

######  "MessageType": 9,

######  "Data": "Some text"

###### }

## Notification about command activation

The notification contains a string with the identifier of a recognized command. After the notification is sent, the system expects the client to perform the action that matches the command.

### Example of a returned value

###### {

######  "MessageType": 10,

######  "Data": "7d4565a7-b408-44d4-afc4-26b96a842446"

###### }

## Notification about activation of enabling/disabling recognition command

**Voice2Med** application provides standard ways of enabling/disabling recognition mode: for example, hot key (double-clicking **Ctrl**) command or voice command. In case **Voice2Med** is connected to by API WebSocket 39999 port (ws://localhost:33999/), Voice2Med does not process such commands and sends them to the external client for processing through this API.

While processing these commands, the client can use its own logic of enabling/disabling recognition.

### Example of an enabling recognition message sent to the client

###### {

######  "MessageType": 12

###### }

### Example of a disabling recognition message sent to the client

###### {

######  "MessageType": 13

###### }

## Notification about changing current recognition mode

Notification about changing current recognition mode comes from the server (**Voice2Med**) to the client (**HIS**) at the moment of changing the recognition mode, for example, when the form metadata is successfully loaded.

### Example of a returned value

###### {

######  "MessageType": 0,

######  "Data": {

######  "Mode": 3,

###### "State": 1,

######  "Info": "a0bc1744-010c-4414-84e6-acabeff7b517"

######  }

###### }

### Description of classes, class fields and special requirements

|  |
| --- |
| **RecognitionModeInfo** |
| Class that contains the description of current recognition mode |
| **Field** | **Field type** | **Requirements** | **Description** |
| Mode | Int32 | Mandatory | Recognition mode:1 – voice activation.2 – continuous recognition.3 – form recognition.4 – form recognition in VoiceWizard mode. |
| Info | String |  | Recognized form identifier |
| State | Int32 |  | 0 – launching.1 – launched and works.2 – stopped. |

## Notification about TTS status

**TTS** (Text-To-Speech) is the component that deals with speech synthesis. It synthesizes the voice confirmation sound of data recognition. Sometimes the server can send notifications about changing the status of the component. These notifications contain information about mistakes (for example, absence of a license for the component), the list of available voices, currently active voice and the speed of speech playback.

### Example of a returned value

###### {

######  "MessageType": 11,

######  "Data": {

######  "VoiceList": [

######  {

######  "Name": "Alexander"

######  }

###### ],

######  "SelectedVoice": {

######  "Name": "Alexander"

######  },

######  "Speed": 1,

######  "State": 0,

######  }

######

###### }

You can also request information about the current status.

### Example of a request to WebSocket

###### {

######  "MessageType": 11

###### }

##### Executing the request through WebSocket can be inconvenient as this request has synchronous logic and WebSocket implies asynchronous communication. Thus, you can get the list of themes through [synchronous request to REST API](#_REST_API). However, it is not mandatory.

## Updating FormSettings

You can change some settings in process of recognition without restarting the process.

This request is used when the form recognition already started and you need to change the settings in process. Otherwise, the request will be ignored. There is no returned synchronized message.

### Example of a request to WebSocket

###### {

######  "MessageType": 10,

######  "Data": {

######  "TtsFieldNameEnabled": true,

######  "TtsFieldValueEnabled": false,

######  "TtsTextFieldValueEnabled": true,

######  "CyclicFocusNotificationEnabled": true,

######  "WizardMode\_TtsFullGroupNamingEnabled": false,

######  "WizardMode\_IsAutoSwitchingEnabled": true,

######  "WizardMode\_IsAutoSwitchingEnabledForText": true

######  }

###### }

# REST API

Executing some requests through WebSocket can be inconvenient as these requests have synchronous logic and WebSocket implies asynchronous communication. Thus, you can use synchronous requests to REST API. It is not mandatory: if you need only one method of transportation, you can still use WebSocket.

## Working with themes (language models)

### Getting the list of themes

**URL:** <http://localhost:39256/api/Theme>

**Method:** GET

**Example of a response:**

###### {

######  "Data": [

######  {

######  "Id": "ct\_2.5.45",

######  "Name": "CT, MRI",

######  "Version": "2.5.45"

######  },

######  {

######  "Id": "main\_2.3.13",

######  "Name": "General",

######  "Version": "2.3.13"

######  },

######  {

######  "Id": "ultrasound\_2.3.22",

######  "Name": "Ultrasound",

######  "Version": "2.3.22"

######  },

######  {

######  "Id": "endoscopy\_2.3.95",

######  "Name": "Endoscopy",

######  "Version": "2.3.95"

######  }

######  ],

######  "Error": null

###### }

## Working with catalogs

### Getting the list

**URL:** <http://localhost:39256/api/Handbook>

**Method:** GET

**Parameters:**

* **showDeleted:** type bool. **GET** parameter. **False** by default. Shows whether to return values marked as deleted.

**Example of a response:**

###### {

######  "Data": [

######  {

######  "Id": "38d61a2b-cde8-4d17-b806-60c5fa8c8769",

######  "Name": "Sex",

######  "IsDeleted": false,

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "Male"

######  },

######  {

######  "Key": "1",

######  "Value": "Female"

######  }

######  ]

######  },

######  {

######  "Id": "bd58aa52-ebb9-4add-ab55-860c1b53d1c4",

######  "Name": "One more catalog",

######  "IsDeleted": true,

######  "AvailableValues": [

######  {

######  "Key": "0",

######  "Value": "First value"

######  },

######  {

######  "Key": "1",

######  "Value": "Second value"

######  }

######  ]

######  }

######  ],

######  "Error": null

###### }

### Deleting the catalog

**URL:** <http://localhost:39256/api/Handbook>

**Method:** DELETE

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a deleted record.

**Example of a response:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and marked as deleted. **False** value, if the record with this id was not found.

### Catalogue recovery

**URL:** <http://localhost:39256/api/Handbook/Restore>

**Method:** POST

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a recovered record.

**Example of a request:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and recovered. **False** value, if the record with this id was not found.

## Working with forms

### Getting the list

**URL:** <http://localhost:39256/api/FromMetadata>

**Method:** GET

**Parameters:**

* **showDeleted: bool** type. **GET** parameter. **False** by default. Shows whether to return values marked as deleted.

**Example of a response (this example contains only one form without child elements)**

###### {

######  "Data": [

######  {

######  "ThemeId": null,

######  "Mode": 1,

######  "Key": "1c0202ad-6c5a-44bd-b963-25d9ce8e0bfe",

######  "Childs": [ ... ],

######  "CustomCommands": [

######  {

######  "Key": "Previous\_field",

######  "Value": "Previous field"

######  },

######  {

######  "Key": "Clear\_field",

######  "Value": "Clear field"

######  },

######  {

######  "Key": "Next\_field",

######  "Value": "Next field"

######  }

######  ],

######  "CultureCode": "ru-RU",

######  "FormSettings": {

######  "TtsFieldNameEnabled": true,

######  "TtsFieldValueEnabled": true,

######  "TtsTextFieldValueEnabled": true,

######  "CyclicFocusNotificationEnabled": true,

######  "WizardMode\_TtsFullGroupNamingEnabled": false,

######  "WizardMode\_IsAutoSwitchingEnabled": true,

######  "WizardMode\_IsAutoSwitchingEnabledForText": true

######  },

######  "Name": "Form №1",

######  "IsDeleted": false

######  }

######  ],

######  "Error": null

###### }

### Deleting

**URL:** <http://localhost:39256/api/FromMetadata>

**Method:** DELETE

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a deleted record.

**Example of a response:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and marked for deletion. **False** value, if the record with this id was not found.

### Recovery

**URL:** <http://localhost:39256/api/FromMetadata/Restore>

**Method:** POST

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a recovered record.

**Example of a response:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and recovered. **False** value, if the record with this id was not found.

## Working with TTS

### Getting the current status of TTS

**URL:** <http://localhost:39256/api/Tts>

**Method:** GET

**Example of a response:**

###### {

######  "Data": {

######  "VoiceList": [ { "Name": "Alexander" } ],

######  "SelectedVoice": { "Name": "Alexander" },

######  "Speed": 1,

######  "State": 0

######  },

######  "Error": null

###### }

|  |
| --- |
| **TtsStateInfo** |
| **Field** | **Field type** | **Description** |
| VoiceList | List of voices |  |
| SelectedVoice | Chosen voice |  |
| Speed | Decimal | Speed of playback |
| State | Int32 | Status:0 - normal functioning.1 - internal error, it is possible that TTS was installed incorrectly.2 - license check error. |

## Working with field types

### Getting the list of available types

**URL:** <http://localhost:39256/api/FormElementType>

**Method:** GET

**Parameters:**

* **showDeleted: bool** type. **GET** parameter. **False** by default. Shows whether it is needed to return values marked as deleted.

**Example of a response**

###### {

######  "Data": [

######  {

######  "Id": 1,

######  "Name": "Boolean",

######  "IsDeleted": false

######  },

######  {

######  "Id": 2,

######  "Name": "Integer",

######  "IsDeleted": false

######  },

######  {

######  "Id": 3,

######  "Name": "Decimal",

######  "IsDeleted": false

######  },

######  {

######  "Id": 4,

######  "Name": "Date",

######  "IsDeleted": false

######  },

######  {

######  "Id": 5,

######  "Name": "Time",

######  "IsDeleted": false

######  },

######  {

######  "Id": 6,

######  "Name": "Date and time",

######  "IsDeleted": false

######  }

######  ],

######  "Error": null

###### }

### Adding custom fields

**URL:** <http://localhost:39256/api/FormElementType>

**Method:** POST

**Parameters are sent in Body:**

###### {

######  "Id": 465,

######  "Name": "MyNewType",

######  "XmlGrammar": "<?xml version=\"1.0\" encoding=\"utf-8\"?> <grammar xml:lang=\"ru-RU\" root=\"Sex\" mode=\"voice\" version=\"1.0\" xmlns=\"http://www.w3.org/2001/06/grammar\" tag-format=\"semantics/1.0\"> <rule id=\"Sex\" scope=\"public\"> <one-of> <item>Male<tag>out='M.'</tag></item> <item>Female<tag>out='F.'</tag></item> </one-of></rule></grammar>"

###### }

**Id -** int; identifier, more than 1000.

**Name -** string; name.

**XmlGrammar -** string; String with SRGS grammar. In tags, **out** variable will return the resulting recognized value.

##### Building SRGS grammar for the custom type is not a standard task. However, an experienced specialist can build the grammar according to the following standards:

##### 1) SRGS grammar: <https://www.w3.org/TR/speech-grammar/>;

##### 2) SISR semantic interpretation: <https://www.w3.org/TR/semantic-interpretation/>.

##### The tags format of semantic interpretation is limited to semantic/1.0 format.

**Example of a response:**

###### {

######  "Data": [

######  {

######  "Id": 1,

######  "Name": "Boolean",

######  "IsDeleted": false

######  },

######  {

######  "Id": 2,

######  "Name": "Integer",

######  "IsDeleted": false

######  },

######  {

######  "Id": 3,

######  "Name": "Decimal",

######  "IsDeleted": false

######  },

######  {

######  "Id": 4,

######  "Name": "Date",

######  "IsDeleted": false

######  },

######  {

######  "Id": 5,

######  "Name": "Time",

######  "IsDeleted": false

######  },

######  {

######  "Id": 6,

######  "Name": "Date and tim",

######  "IsDeleted": false

######  }

######  ],

######  "Error": null

###### }

### Deleting

**URL:** <http://localhost:39256/api/FormElementType>

**Method:** DELETE

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a deleted record.

**Example of a response:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and marked for deletion. **False** value, if the record with this id was not found.

###  Recovery

**URL:** [http://localhost:39256/api/FormElementType/Restore](http://localhost:39256/api/FromMetadata/Restore)

**Method:** POST

**Parameters:**

* **id: string** type. **GET** parameter. Identifier of a recovered record.

**Example of a response:**

###### {

###### "Data": false,

######  "Error": null

###### }

Data parameter returns **true** value, if the record was found and recovered. **False** value, if the record with this id was not found.

001-300919