SOFTWARE

CHROMATEC ANALYTIC

BUILD 4.0

USER MANUAL

214.00045-51I

REST API INTEGRATION

EXPORT DATA TO LIMS

Revision 10.12.2025

TABLE OF CONTENTS

1 Introduction 5
2 Instrument control 6
2.1 Authorization 6
2.1.1 Receiving basic tokens 6
2.1.2 Obtaining a token for the device management service 7
2.1.3 Refresh tokens 7
2.1.4 Getting a list of connections 8
2.2 Connection to the device 9
2.3 Instrument method 10
2.3.1 Getting instrumental methods 10
2.3.2 Loading an instrumental method 10
2.3.3 Saving an instrumental methods 10
2.3.4 Getting start channels 11
2.3.5 Getting a list of instrumental method parameters 11
2.3.6 Updating an instrument method parameter 13
2.3.7 Instrument method run 14
2.4 Scheduler 14
2.4.1 Scheduler runs 14
2.4.2 Scheduler stops 14
2.4.3 Stopping the current analysis 14
2.4.4 Getting a list of saved sequences for the Scheduler 15
2.4.5 Saving the current method for the Scheduler 15
2.4.6 Loading the current method for the Scheduler 15
2.5 Request chromatograms 16
2.5.1 Request chromatogram ID 16
2.5.2 Request channels 16
2.5.3 Request channel data 18
3 Getting measurement results 19
3.1 What is LIMS? 19
3.2 How do LIMS and Chromatec Analytic relate? 19
3.3 Which LIMS does Chromatec Analytic support? 19
3.4 Barcode Scanner 19
3.5 Exporting data to LIMS 20
3.6 Getting results 21
3.6.1 Requesting the latest result for the chromatograph 21
3.6.2 Requesting multiple results 22

3.6.3 Processing of the received data 23
3.7 Calculation parameters 24
3.7.1 Calculation according to GOST 31369-2021 24
3.7.2 Calculation according to GOST 31371.7-2020 26

1 INTRODUCTION

This guide is intended for third-party software developers who wish to interact with Chromatec Analytic 4 software using the REST API.

API (Application Programming Interface) is a set of methods and rules by which various programs communicate with each other and exchange data.

A REST API is an architectural approach that sets the limits for APIs: how they should be structured and what functionality they should support. This allows you to standardize the operation of software interfaces, making them more convenient and productive.

The word REST is an acronym for Representational State Transfer, which translates into Russian as “representational state transfer”, “representational state transfer” or “self-describing state transfer”.

In Chromatec Analyst 4 using the REST API you can:

- Get a list of connections.
- Receive a list of instrumental methods, transfer methods to the chromatograph.
- Start and stop the analysis sequence (plan).
- Get measurement results (latest or in a specified time interval).

In all requests in this document, the {{host}} variable denotes the base URL of the Chromatec Analytics server.

For example:

```
http://192.168.0.243
```

2 INSTRUMENT CONTROL

2.1 Authorization

2.1.1 Receiving basic tokens

To manage chromatographs via the API, you need to authorize and receive an authorization token, which will be subsequently transmitted in the headers of POST requests.

In the body of the POST request, you need to pass an object with the administrator login and password (see code below). The request is made to the endpoint:

```
POST {{host}}/api/auth/v1/login

{
  "login": "admin",
  "password": "admin"
}
```

The default login and password are listed here. In your case they may be different if you changed them.

The response is an object of the following structure:

```
{
  "userId": "admin",
  "roleId": "admin",
  "permissions": [
    "admin",
    "all_data_access",
    "modify_connections",
    "modify_projects",
    "update_firmware",
    "modify_config",
    "modify_sequence",
    "modify_method_instr",
    "modify_method_proc",
    "change_passport_values",
    "change_passport_keys",
    "change_shewhart_data",
    "modify_channels",
    "manual_markup",
    "modify_chrom_method",
    "modify_retention",
    "delete_data"
  ]
}
```

We are not interested in this object. The necessary data must be extracted from the response headers.

The Authorization header contains the {{Access-Token}} value we are looking for. Its lifetime is 5 minutes.

You also need to get the value of the Refresh-Token header. Its lifespan is 1 month.

{{Access-Token}} must be updated regularly.

Here and below, the parameters in brackets {{}} indicate that these are header values.

Example {{Access-Token}}

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJhZG1pbiIsInJvbGVJZCI6ImFkbWluIiwicGVybWlzc2lvbnMiOlsiYWRtaW4iLCJhbGxfZGF0YV9hY2Nlc3MiLCJtb2RpZnlfY29ubmVjdGlvbnMiLCJtb2RpZnlfcHJvamVjdHMiLCJ1cGRhdGVfZmlybXdhcmUiLCJtb2RpZnlfY29uZmlnIiwibW9kaWZ5X3NlcXVlbmNlIiwibW9kaWZ5X21ldGhvZF9pbnN0ciIsIm1vZGlmeV9tZXRob2RfcHJvYyIsImNoYW5nZV9wYXNzcG9ydF92YWx1ZXMiLCJjaGFuZ2VfcGFzc3BvcnRfa2V5cyIsImNoYW5nZV9zaGV3aGFydF9kYXRhIiwW9kaWZ5X2NoYW5uZWxzIiwibWFudWFsX21hcmt1cCIsIm1vZGlmeV9jaHJvbV9tZXRob2QiLCJtb2RpZnlfcmVmcmVzaF90b2tlbiIsImZGVZRlX2RhdGEiXSwibG9naW5JZCI6IlpUeVEtLWgwU05pcV96cklwYkxrUEEiLCJkZXZpY2UiOiIxOTIuMTY4LjAuODMiLCJleHAiOjEMDyMjExODUsIm5iZiI6MTcwMDIyMDg4NX0.jZgl68Ob1jN4fOwiZv2CziJYpq5PU_18-gcLyhLK2QQ
```

Example {{Refresh-Token}}

```
ZTyQ--h0SNiq_zrIpbLkPA.4K8T9aVNRCKT7okE2XL78g.gYD8vgcwTXKYkn4JWSAd6R9vVQhtSoWaSjqebwgyDBg
```

2.1.2 Obtaining a token for the device management service

Next, you should get another token by requesting the endpoint:

```
POST {{host_auth}}/api/auth/v1/access
Authorization: {{Access_Token}}
```

Here the {{Access-Token}} in the request header is obtained in the previous step.

In the response, extract the value {{Refresh-Token-Instrument}} from the Refresh-Token header.

This is a special token for the device management service. Note that this is also retrieved from the Refresh-Token header, but it is not the same as the {{Refresh-Token}} obtained in the previous step.

This way you will receive three tokens:

- {{Access-Token}}
- {{Refresh-Token}}
- {{Refresh-Token-Instrument}}

We will need the latter once, when connecting to the device.

2.1.3 Refresh tokens

As already mentioned, {{Access-Token}} lives for 5 minutes and needs to be renewed regularly.

It is not recommended to receive {{Access-Token}} and {{Refresh-Token}} every time by request to the /login endpoint, since in this case the login and password are always transmitted in clear text, and also increases the base of stored {{Refresh-Token}}.

To refresh an {{Access-Token}}, use a POST request to the /refresh endpoint with the resulting long-lived {{Refresh-Token}} in the header.

```
POST {{host}}/api/auth/v1/refresh
Refresh-Token: {{Refresh-Token}}
```

In the response, extract the {{Access-Token}} value from the Authorization header again, as well as the new {{Refresh-Token}} value from the Refresh-Token header.

Next, you should make a POST request from the **/confirm** endpoint:

```
POST {{host}}/api/auth/v1/confirm
Refresh-Token: {{Refresh-Token}}
```

This request will destroy the previous Refresh-Token. The new one will become relevant.

2.1.4 Getting a list of connections

The request is made to the endpoint:

```
GET {{host}}/api/p4/v1/5000/profileManager/profiles
```

Example:
https://a4.chromatec.ru/api/p4/v1/5000/profileManager/profiles

The response is an object of the following structure:


```
[
  {
    "id": "fb273a5767a04e4cbd59197d492702a3",
    "name": "2052411",
    "startChannelsCount": 1,
    "autoRun": false,
    "diagnosticStorageTime": 15,
    "instruments": [
      {
        "pluginId": "PM3",
        "name": "ПМ3",
        "instrumentSettings": null,
        "connectionSettings": "<?xml ... >",
        "connectionParameters": {
          "connectionType": 0,
          "port": 2010,
          "address": 1531816128,
          "comPort": 0,
          "msdAddress": 0,
          "msdPort": 0,
          "isMsdOn": false,
          "serial": null,
          "deviceType": 0,
          "pluginId": "",
          "serialPortBaudRate": 0,
          "localEndpoint": null,
          "isConnected": false,
          "isUnreacheable": false,
          "cts": false
        },
        "startChannel": 0,
        "isActive": true,
        "id": "837a1d7e-4f99-c76d-f646-5ecd63027a3e",
        "index": 1,
        "serialNumber": "2052411",
        "version": "v 03.21.17.759"
      },
      .....
    ],
    "sessions": [
      {
        "id": "fa4c6ebadaf445019bada3aeaa21e4c7",
        "name": null,
        "startChannel": 0,
        "project": null,
        "method": null,
        "synchronizeTime": false,
        "analysisTime": 0,
```

```
        "channelsSettings": null,
        "isDynamic": false
      }
    ],
    "method": null,
    "autoLoadMethod": false,
    "autoSaveDiagnosticsLog": false,
    "remoteAddress": null,
    "remotePort": 0,
    "isConnected": false,
    "userId": "a.demakov"
  },
  .....
]
```

For further work, you will need to extract from this array for each connection its id - a unique identifier.

2.2 Connection to the device

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{host}}/api/p4/v1/5000/profileManager/connect?profileId={{id}}&token={{Refresh-Token-Instrument}}
Authorization: {{Access-Token}}
```

If any token's lifetime has expired, you will receive a message:

```
Bad token
```

If the connection is successful, it will return a response with an HTTP status of 200 OK, as well as your address of the local instrument server you connected to.

For example, http://192.168.0.83:5002

In all subsequent requests to control the chromatograph, the local server address should be used as the host. It will be described as the {{local}} variable.

2.3 Instrument method

2.3.1 Getting instrumental methods

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
GET {{local}}/aggregatorRest/instrumentMethods
Authorization: {{Access-Token}}
```

В ответе вы получите массив названий методов. Например:

```
[
  "test_instr_method"
]
```

2.3.2 Loading an instrumental method

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/aggregatorRest/loadMethod?methodName={{methodName}}
Authorization: {{Access-Token}}
```

{{methodName}} – имя метода, который вы хотите загрузить.

Response:

```
true
```

2.3.3 Saving an instrumental methods

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/aggregatorRest/saveMethod?methodName={{methodName}}
Authorization: {{Access-Token}}
```

{{methodName}} – the name of the method you want to save.

Response:

```
true
```

2.3.4 Getting start channels

```
POST {{local}}/aggregatorRest/getStartChannels
Authorization: {{Access-Token}}
```

In the response you will receive an array of data on the start channels and the current state of the channels (stage and time):

```
[
  {
    "stage": 4,
    "time": 4.583333333333333,
    "analysisTime": 300,
    "number": 1
  }
]
```

Chromatograph stages (parameter "stage"):

0. Idle	5. Preparing for blowing	10. Fatal error
1.Not ready	6. Blowing	11. Run method
2.Waiting	7. Gas saving	12. Analysis start
3. Ignition	8. Cooling	13. Waiting for AUX
4. Ready	9. Analysis	

2.3.5 Getting a list of instrumental method parameters

Refresh {{Access-Token}} and {{Refresh-Token}}

```
GET {{local}}/aggregatorRest/params?deviceContext={{deviceContext}}&paramContext={{paramContext}}
Authorization: {{Access-Token}}
```

{{deviceContext}} – Device context		**{{paramContext}} – Parameter context**	
Bit mask that allows you to filter the list of received parameters by device category		Bit mask that allows you to filter the list of received parameters by parameter category	
0x0001	Unknown device settings	**0x0001**	Parameters without category
0x0002	Chromatograph parameters	**0x0002**	Configuration parameters
0x0004	Gas and power module parameters	**0x0004**	Expert configuration parameters
0x0008	Start channel parameters	**0x0008**	Status parameters
0x0010	Thermostat parameters	**0x0010**	Information parameters (firmware versions)
0x0020	Sampler parameters	**0x0020**	Session parameters (recorded signals)
0x0040	Transferline parameters	**0x0040**	Diagnostic parameters
0x0080	Pump parameters	**0x0080**	Messages
0x0100	Valve parameters	**0x0100**	Instrument method parameters
0x0200	Solenoin valve parameters	**0x0200**	Ignition parameters
0x0400	Inlet parameters	**0x0400**	Cooling parameters
0x0800	Column parameters	**0x0800**	Groupable parameters

0x1000	Oven parameters	**0x1000**	Purge parameters
0x2000	Gas regulator parameters	**0x2000**	Accident reports
0x4000	Parameters of auxiliary gas regulators and meters	**0xFFFF**	All parameters
0x8000	Methanizer parameters		
0x10000	Detector parameters		
0x20000	Parameters of external detectors		
0x40000	Accessory settings		
0xFFFFF	All devices settings		

The response returns an array of objects:


```
[
  {
    "name": "AdditionalChannel1",
    "deviceName": "Силовой модуль",
    "instrumentName": "ПМ2",
    "id": "AdditionalChannel1",
    "deviceId": "TemperatureModule",
    "instrumentId": "3a0c2230-babd-500c-8005-056e1556463d",
    "instrumentIndex": 1,
    "deviceNumber": 1,
    "wrong": false,
    "measured": 15.0141,
    "controlled": null,
    "assigned": null,
    "progNames": null,
    "enumNames": null,
    "orderNumber": 0,
    "paramCategory": 1,
    "deviceCategory": 1,
    "type": "",
    "group": null,
    "range": null,
    "range1": null,
    "range2": null,
    "comment": null,
    "deviceComment": null,
    "allowableValue": null,
    "device": {
      "isOutOfNorm": false,
      "comment": null,
      "name": "Силовой модуль",
      "id": "TemperatureModule",
      "number": 1,
      "isNotConnected": false
    }
  },
  .....
]
```

2.3.6 Updating an instrument method parameter

Refresh {{Access-Token}} and {{Refresh-Token}}

```
PATCH {{local}}/aggregatorRest/setParam
Authorization: {{Access-Token}}

{
   "name": "AdditionalChannel2",
    "deviceName": "Силовой модуль",
    "instrumentName": "ПМ2",
    "id": "AdditionalChannel1",
    "deviceId": "TemperatureModule",
    "instrumentId": "3a0c2230-babd-500c-8005-056e1556463d",
    "instrumentIndex": 1,
    "deviceNumber": 1,
    "wrong": false,
    "measured": 15.014,
    "controlled": null,
    "assigned": null,
    "progNames": null,
    "enumNames": null,
    "orderNumber": 0,
    "paramCategory": 1,
    "deviceCategory": 1,
    "type": "",
    "group": null,
    "range": null,
    "range1": null,
    "range2": null,
    "comment": null,
    "deviceComment": null,
    "allowableValue": null,
    "device": {
        "isOutOfNorm": false,
        "comment": null,
        "name": "Силовой модуль",
        "id": "TemperatureModule",
        "number": 1,
        "isNotConnected": false
    }
}
```

To call this method, you must pass a {{paramContext}} and a {{deviceContext}} object in the body of the request. This is one of the array objects obtained in the previous step.

2.3.7 Instrument method run

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/aggregatorRest/runMethod?method
Authorization: {{Access-Token}}
Content-Type: application/json

{
  "startChannel": 2
}
```

{{methodName}} – the name of the method you want to save.

The body of the request contains JSON with the start channel number.

If you pass start channel = 0, then the method will be applied to all channels.

If you incorrectly specified the start channel, or such a channel does not exist, it will generate the error "No ActiveAggregator".

2.4 Scheduler

2.4.1 Scheduler runs

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/scheduler/start
Authorization: {{Access-Token}}
```

2.4.2 Scheduler stops

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/scheduler/stop
Authorization: {{Access-Token}}
```

2.4.3 Stopping the current analysis

Stopping the Scheduler (sequence) will not stop the current analysis. It will be completed to the end. To forcefully abort it:

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/stop
Authorization: {{Access-Token}}
```

2.4.4 Getting a list of saved sequences for the Scheduler

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
GET {{local}}/scheduler/plans
Authorization: {{Access-Token}}
```

Ответ:

```
[
  "",
  "exmapleName"
]
```

2.4.5 Saving the current method for the Scheduler

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/scheduler/save?name={{methodName}}
Authorization: {{Access-Token}}
```

{{methodName}} – the name of the method you want to save.

2.4.6 Loading the current method for the Scheduler

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
POST {{local}}/scheduler/load?name={{methodName}}
Authorization: {{Access-Token}}
```

{{methodName}} – the name of the method you want to save.

2.5 Request chromatograms

2.5.1 Request chromatogram ID

Refresh {{Access-Token}} and {{Refresh-Token}}

Next run the request:

```
GET {{local}}/analyses
Authorization: {{Access-Token}}
```

In the response you will receive an array of chromatogram identifiers. For example:

```
[
  "0dd2016952e84b2e94034bb7f9154231",
  "0f913a09989e45c9b89b664ee3ba7fd8",
  "20ec63db42c0458ba3f691a907ab1f1f"
]
```

2.5.2 Request channels

A channel in Chromatec Analytic software is an object that contains not only “raw” chromatographic analysis data, but also additional information necessary for constructing and processing a chromatogram.

Refresh {{Access-Token}} and {{Refresh-Token}}

Request chromatogram ID. ID of interest – {{ChromatogramId}}

Next run the request:

```
GET {{local}}/channels?chromatogramId={{ChromatogramId}}
Authorization: {{Access-Token}}
```

In the response you will receive an array of channels - A4Channel objects:

```
class A4Channel {
  /// Channel ID
  id: string;
  /// Channel name
  name: string;
  /// This channel is integrable?
  is_integrable: boolean | undefined;
  /// This chennel is permanent? (cannot be deleted)
  is_permanent: boolean | undefined;
  /// Have the peaks been edited on the channel?
  is_peak_points_edited: boolean | undefined;
  /// Channel information
  info: A4ChannelInfo;
  /// RAW data information
  samples_info: A4SamplesInfo;
  /// RAW data
  samples_data: number[] | undefined;
  /// RAW data (X-axis)
  samples_data_x: number[] | undefined;
}

/// Channel information
class A4ChannelInfo{
  /// Information about the detector (components will be compared using this data)
```

```
  Detector: A4DetectorInfo;
  /// Information
  q3BuildInfo: A4MSDQ3BuildInfo;
  /// Original ID
  original_id: string;
}

/// Detector information
class A4DetectorInfo{
  /// Detector name
  name: string;
  /// Additional data. For example – detector ordinal number (TCD-2)
  adds: string;
  /// ID of the component associated with the detector
  componentId: string;
  /// Name of the component associated with the detector
  /// (needed to form a name)
  componentName: string;
  /// Number of the mass on which this channel is built
  massNum: number;
  /// The mass on which this channel is built
  mass: double;
}

/// Information about building this channel from MSD-Q3 data
class A4MSDQ3BuildInfo {
  /// The ordinal number of the component in the method
  componentIdx: number;
  /// Component name
  componentName: string;
  /// Number of the mass on which this channel is built
  massNum: number;
  /// The mass on which the channel is built
  mass: number;
  /// Precursor mass
  preMass: number;
}

/// RAW data information
class A4SamplesInfo {
  /// Start time
  start_time: number | undefined;
  /// Frequency, Hz
  frequency: number | undefined;
  /// Response unit
  response_unit: string;
  /// Additional channel parameters
  extra: A4SamplesInfoExtra;
}

/// Additional channel parameters
class A4SamplesInfoExtra : A4Extra {
  /// The channel has X-coordination data (retention time)
  haveXValues: boolean;
  /// Identifiers of scans based on which the channel was built
  msdScanIds: number[];
}
```

2.5.3 Request channel data

Channel data (also "raw data") – an array of Y coordinates of the chromatogram, detector signal values in mV. Can be diagnostic - values of temperatures, flow rates or gas pressures.

The X coordinate is not recorded in the detector data; it is calculated based on the frequency value and analysis start time (see A4SamplesInfo).

Refresh {{Access-Token}} and {{Refresh-Token}}

Request channels.

For the channel you are interested in, get the identifier – {{ChannelId}}.

You can receive data not only after the analysis is completed, but also dynamically. The {{Offset}} parameter is used for this. Each time you request channel data, you will receive an array of numbers in response. All data received at the time of the request is displayed. If you want to query data by timer, you must calculate a new {{Offset}}, the index of the last element of the array, each time. On the first request {{Offset}} = 0.

Next run the request:

```
GET {{local}}/samples?chromatogramId={{ChromatogramId}}&channelId={{ChannelId}}&offset={{Offset}}
Authorization: {{Access-Token}}
```

Request example:

```
[
  1.12345,
  1.12344,
  1.12323,
  ...
]
```

3 GETTING MEASUREMENT RESULTS

3.1 What is LIMS?

LIMS – Laboratory Information Management.

LIMS is a special software designed to manage laboratory work flows and documents. It optimizes the collection, analysis, return and reporting of laboratory data. It is often used in the pharmaceutical and food industries.

The purpose of LIMS is to obtain reliable information based on test results and optimize the management of this information in order to use it to make correct timely management decisions.

3.2 How do LIMS and Chromatec Analytic relate?

The chromatograph is one of the sources of information about manufactured products or processes occurring in production, along with other analytical equipment and other suppliers of information necessary for product quality management.

Chromatec Analytic transmits to LIMS the results of chromatographic analysis, as well as special calculations according to some regulatory documents.

3.3 Which LIMS does Chromatec Analytic support?

Chromatec Analytic does not support any specific types of LIMS. It is more correct to say that it supports any kind of LIMS, since almost all of them have flexible settings that allow you to receive data from external sources.

Chromatec Analytic allows you to export data in several formats.

3.4 Barcode Scanner

When working with LIMS, a barcode reader (scanner) can be used to automate filling in sample information. This device is connected to a computer via USB or PS/2 interface. Recognized by Windows automatically. No configuration is required.

To read the information from the barcode at the start of a new analysis (or when filling out the passport before the analysis), you need to put the cursor in the Name of the sample field of the chromatogram passport and bring the reader to the barcode. The sample ID will be automatically entered in this field and will be included in the uploaded file when exporting.

3.5 Exporting data to LIMS

Currently, there are two possible ways to provide data in Chromatec Analytic 4.0:

- Data transfer via REST (GET request).
- Data transmission via industrial ModBus protocol.

The advantages of using ModBus are the feedback, i.e. Chromatec Analytic not only transmits data to LIMS, but can also receive control commands in response, for example, a command to load a certain mode into the chromatograph.

The description of the Modbus data is given in the corresponding manual.

Configuring the export to LIMS is reduced to the usual report setup .

In the main settings, you need to select JSON as the report format.

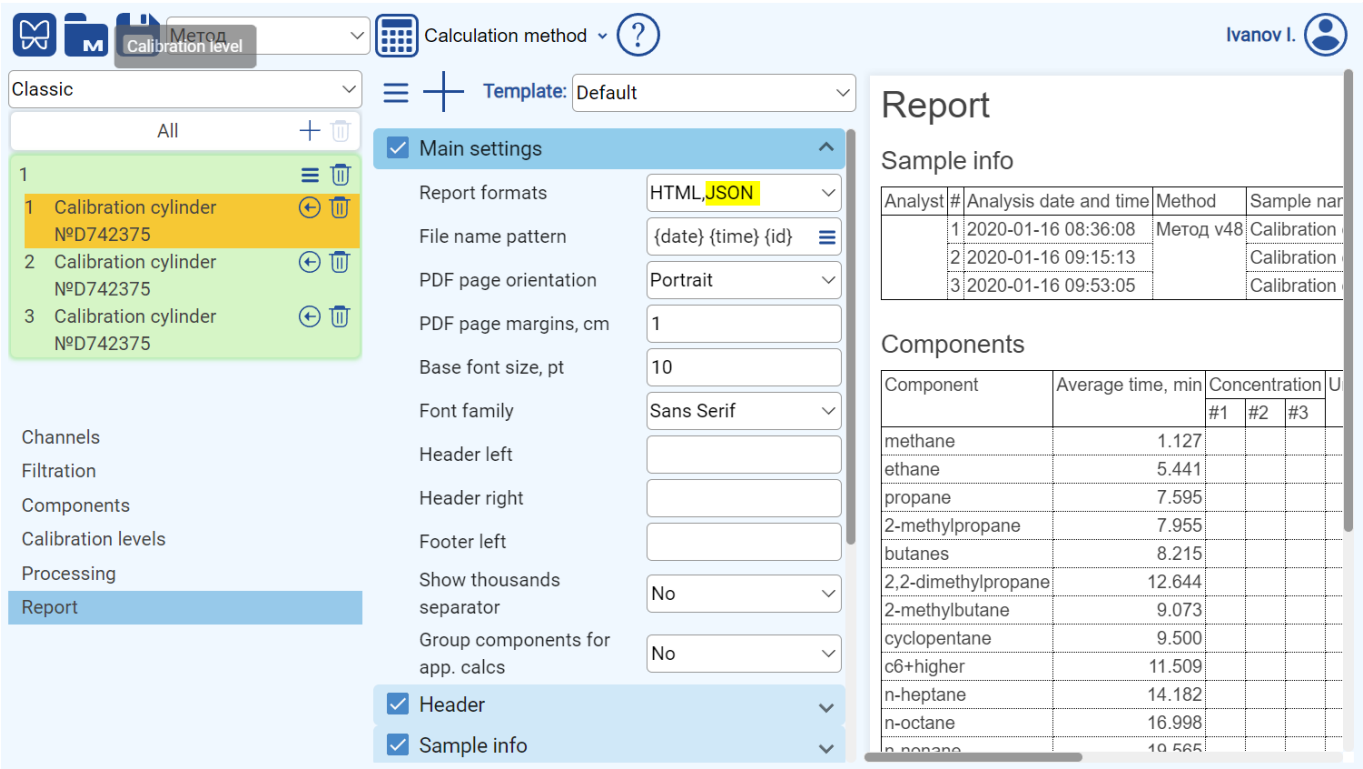


Fig 3.1 Selection of the report format - JSON

3.6 Getting results

LIMS can request analysis results using the REST API. There are two types of GET request available.

Requests are made to the endpoint:

```
{host}/api/a4r/v1/report/store
```

For example:

```
http://a4.chromatec.ru/api/a4r/v1/report/store
```

3.6.1 Requesting the latest result for the chromatograph

To fulfill the request, you need to know the serial number of the chromatograph. It is specified as a parameter **serial_number**:

```
{host}/api/a4r/v1/report/store?serial_number=123456
```

The answer is an array of objects ExportData:

```
class ExportData {
    id: string;
    name: string;
    tables: ExportTable[];
}
class ExportTable {
    id: string;
    name: string;
    params: ExportParam[];
}
class ExportParam {
    key: string;
    name: string;
    value: any;
}
```

Example:

```
[
    {
        "id": "gost31371_7_2020",
        "name": "GOST 31371.7-2020",
        "tables": [...]
    },
    {
        "id": "gost3169_2021",
        "name": "GOST 31369-2021",
        "tables": [...]
    },
    {
        "id": "gost57975_1_2017",
        "name": "GOST 57975.1-2017",
        "tables": [...]
    },
```

]

3.6.2 Requesting multiple results

You can request multiple results at once.

To do this, pass the name of the project and (optionally) the start and end dates of the time interval of interest as a parameter.

Parameters:

project Project Name

from_time Start of the time interval (optional parameter)

to_time End of time interval (optional parameter)

For example, for the GOST 31371 project, we want to receive reports for the period from 2023-08-08 10:00:00 to 2023-08-08 11:00:00:

```
{host}/api/a4r/v1/report/store?project=ГОСТ 31371&from_time=2023-08-08 10:00:00&to_time=2023-08-08
12:00:00
```

The answer is an array of the following structure:

```
interface IExportResults {
   path: string;
   dateCreation: Date;
   results: ExportData[];
}
class ExportData {
   id: string;
   name: string;
   tables: ExportTable[];
}
class ExportTable {
   id: string;
   name: string;
   params: ExportParam[];
}
class ExportParam {
   key: string;
   name: string;
   value: any;
}
```

Example:

```
[
   {
      "path": "C:\\Source\\analytic4\\data\\reports\\analytic\\ГОСТ 31371\\2023-08-08  10-23-17
1,2,3.json",
      "dateCreation": "2023-08-08T08:13:18.251Z",
      "results": [
         {
            "id": "gost31371_7_2020",
            "name": "GOST 31371.7-2020",
            "tables": [...]
         },
         {
            "id": "gost3169_2021",
            "name": "GOST 31369-2021",
            "tables": [...]
         },
      ]
   },
   .....
```

```
]
```

3.6.3 Processing of the received data

As you can see from the above description, the export result is an array of objects, each of which has a unique value of the "id" property.

The "name" property contains the localized name of the calculation.

The value of the "tables" property is an array of tables, each of which has a unique identifier value "id" and an array of parameters.

For each parameter, the "name" property can contain the name of the component, or an additional identifier, see the figure below.

For example, in order to get the concentrations of all components from the **gas_composition** array, you need to select all the elements that have **key == conc**.

To find all the parameters of methane, you need to get all the elements with **name == methane** from the **gas_composition** array.

Etc.

The parameters of the various calculations are listed in the next section.

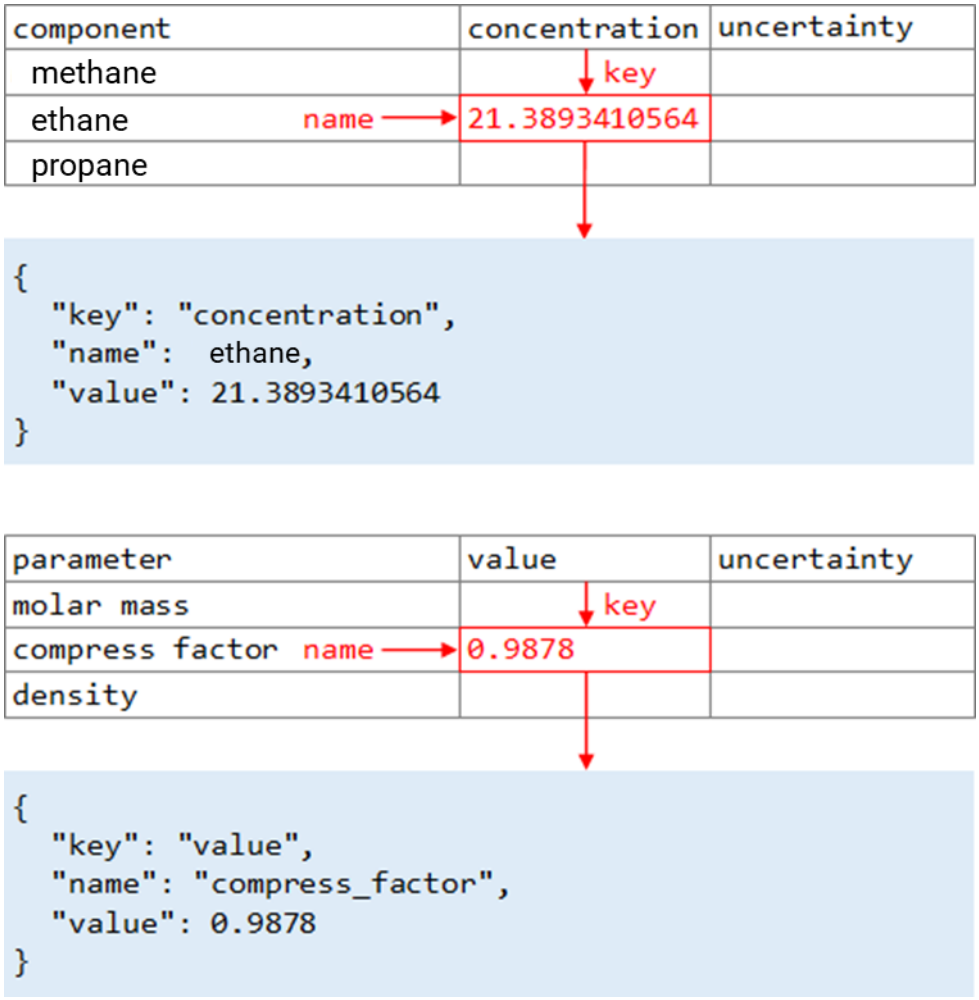


Fig 3.2 – The principle of forming an ExportParam object

3.7 Calculation parameters

3.7.1 Calculation according to GOST 31369-2021

```
{
    "id": "gost31369_2021",
    "name": "GOST 31369-2021",
    "tables": [...]
}
```

Normalized input data

```
{
    "id": "normalized_conc",
    "name": "Input normalized fractions ",
    "params": [{
            "key": "units.molar_percent",
            "name": "ethane",
            "value": 21.389341056409574
        },
        .....
    ]
}
```

The value of the "name" parameter is the name of the component.

Possible values of the "key" parameter:

Значение	Тип данных	Описание
molar_percent	number	Molar fraction, %
mass_percent	number	Molar fraction, %
volume_percent	number	Volume fraction, %
molar_uncert	number	Standard uncertainty of the molar fraction, %
molar_uncert_ext	number	Extended uncertainty of the molar fraction, %

Physico-chemical parameters of an ideal gas

```
{
    "id": "ideal_gas",
    "name": "Physico-chemical parameters of an ideal gas",
    "params": [
        {
            "key": "physical_parameter_value",
            "name": "molar_mass@g/mol",
            "value": 38.76
        },
        .....
    ]
}
```

Possible parameter values:

key	Type	name	Description

physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	molar_mass@g/mol	Molar mass, g/mol
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	compress_factor	Compressibility factor
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	high_combustion_heat_molar@kJ/mol	Molar heat of combustion higher, kJ/mol
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	low_combustion_heat_molar@kJ/mol	The molar heat of combustion is the lowest, kJ/mol
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	high_combustion_heat_mass@MJ/kg	The mass heat of combustion is higher, MJ/kg
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	low_combustion_heat_mass@MJ/kg	The mass heat of combustion is the lowest, MJ/kg
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	high_combustion_heat_volume@MJ/m3	The volumetric heat of combustion is higher, MJ/m^3
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	low_combustion_heat_volume@MJ/m3	The volumetric heat of combustion is the lowest, MJ/m^3
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	density_abs@kg/m3	Density, kg/m^3
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	density_rel	Relative density
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	high_wobbe@MJ/m3	The Wobbe number is higher, MJ/m^3
physical_parameter_value physical_parameter_uncert_ext	Value Uncertainty	number number	low_wobbe@MJ/m3	The Wobbe number is the lowest, MJ/m^3

Physical and chemical parameters of real gas

```
"id": "real_gas",
```

Otherwise, the structure is similar to an ideal gas.

3.7.2 Calculation according to GOST 31371.7-2020

```
{
   "id": "gost31371_7_2020",
   "name": "GOST 31371.7-2020",
   "tables": [...]
}
```

Input non-normalized fractions

```
{
   "id": "input_unnormalized_concentrations",
   "name": " Input non-normalized fractions, molar .%",
   "params": [
      {
         "key": "#4",
         "name": "ethane",
         "value": 0.007505432770666584
      },
      .....
   ]
}
```

The value of the "name" parameter is the name of the component.

Possible values of the "key" parameter:

Value	Data type	Description
#<chromatogram number >	number	Molar fraction, %
range_mol	number	Range, molar %
range_mol_norm	number	Range regulation, molar %
acceptability	boolean	Acceptability

Table of calibration functions

```
{
    "id": "calibration_functions_ab",
    "name": "Table of calibration functions",
    "params": [
        {
            "key": "calibration_function",
            "name": "ethane",
            "value": "X = (2.1808e-4) * A / q"
        },
        .....
    ]
}
```

Acceptability of calibration factors

```
{
    "id": "calibration_factors_acceptability",
    "name": "Acceptability of calibration factors",
    "params": [
        {
            "key": "#1",
            "name": "ethane",
            "value": 0.0002180760478864956
        },
        .....
    ]
}
```

The value of the "name" parameter is the name of the component.

Possible values of the "key" parameter:

	Data type	Description
#<chromatogram number >	number	Calibration factor
average	number	Average calibration factor
calibration_factor_range	number	Range of the calibration factors, rel. %
normative	number	Regulation, rel. %
acceptability	boolean	Acceptability

Verification of reference standard and sample compliance

```
{
    "id": "standard_and_sample_compliance",
    "name": "Verification of reference standard and sample compliance",
    "params": [
        {
            "key": "conc_not_normalized",
            "name": "ethane",
            "value": 4.663504906873109
        },
        .....
    ]
}
```

The value of the "name" parameter is the name of the component.

Possible values of the "key" parameter:

Value	Data type	Description
conc_not_normalized	number	An unnormalized fraction, molar %
min	number	Minimum allowable range of calibration standard, mol.%
max	number	The maximum allowable range of the calibration standard, mol.%
standard_conc	number	Passport value of the molar fraction of standard, %
acceptability	boolean	Acceptability

Normalized sample composition

```
{
    "id": "gas_composition",
    "name": " Normalized sample composition",
    "params": [
        {
            "key": "conc",
            "name": "ethane",
            "value": 21.4
        },
        .....
    ]
}
```

The value of the "name" parameter is the name of the component.

Possible values of the "key" parameter:

Value	Data type	Description
conc	number	Normalized fraction, molar %
uncertainty_mol_ext	number	Extended uncertainty, molar %
analysis_method	number	Method of analysis