



# **New Features in ibaPDA v8.7.0**

2024-01-24

## Table of contents

<b>General remarks .....</b>	<b>3</b>
<b>HTTP(S) Interface .....</b>	<b>4</b>
Interface .....	4
Module .....	4
Request settings .....	6
Response settings.....	14
Analog and digital signals.....	19
Validation warnings .....	21
<b>Multi-state label text rotation .....</b>	<b>22</b>
<b>LMI-Gocator Health module.....</b>	<b>24</b>
<b>Snapshots interface .....</b>	<b>27</b>
<b>FFT view and Cycle view.....</b>	<b>35</b>
Plane sorting based on the value of a process signal .....	35
Scaling of the contour Z-axis based on the value of a process signal .....	35

**General remarks**

None

## HTTP(S) Interface

Starting with version 8.7.0, ibaPDA provides a new interface for sending HTTP(S) requests.

Order Information	
Order no.	31.001018
Name	ibaPDA-Interface-HTTP(S)
License	Unlimited connections / modules

## Interface

The interface overview contains a list of the configured HTTP(S) modules together with statistics for each module.

The statistics include

- Number of sent requests
- Number of errors
- Last response body size (bytes)
- Last response time (ms)
- Minimum response time (ms)
- Maximum response time (ms)

HTTP(S)							DEMO
							<input type="button" value="Open log file"/> <input type="button" value="Reset statistics"/>
Module	Read counter	Error counter	Data size	Response time			
				Actual	Min	Max	
▶ HTTP(S) - Redmine (2)	2	2	0	0 ms	0 ms	0 ms	
HTTP(S) - Weather (6)	1	0	1050	237 ms	237 ms	237 ms	
HTTP(S) - Post (7)	0	0	0	0 ms	0 ms	0 ms	
HTTP(S) - gitlab oauth2 (16)	4	1	2090	73 ms	73 ms	425 ms	
HTTP(S) - Local (9)	2	2	0	0 ms	0 ms	0 ms	
HTTP(S) - teams oauth (17)	4	4	0	0 ms	0 ms	0 ms	
HTTP(S) (18)	0	0	0	0 ms	0 ms	0 ms	

The **Reset statistics** button can be used to reset the statistics of all modules.

**Open log file** opens the log file of the HTTP(S) interface (if available).

## Module

HTTP(S) modules provides functionality to configure the request and to handle the response. Each module contains exactly one request.

## General properties

The property grid of the HTTP(S) module provides numerous settings:

## HTTP(S) (19)

RE  
ST **General**

RE  
ST Request settings

~ Analog

⏏ Digital

<b>Basic</b>	
Module Type	HTTP(S)
Locked	None
Enabled	True
Name	<b>HTTP(S)</b>
Comment	
Module No.	<b>19</b>
Timebase	<b>10 ms</b>
Use module name as prefix	False
<b>Diagnostics</b>	
Verbose logging	False
<b>Processing</b>	
Decimal point	<b>Point</b>
<b>Trigger</b>	
Update time	5000 ms
Send mode	On trigger
Trigger signal	<b>[4.0] send http request</b>

### Verbose logging

If set to false, only errors are logged. Otherwise, all request and response details are logged.

### Decimal point

Configures which character (point or comma) is used as the decimal point.

### Update time

Defines the delay between two consecutive requests depending on the send mode.

- Cyclic: A request is sent periodically based on the update time
- On change/On trigger: The update time acts as a dead time

### Send mode

Defines when a new HTTP(S) request is sent.

- Cyclic: A request is sent periodically based on the update time
- On change: A request is sent each time the data of the trigger signal changes
- On trigger: A request is sent each time a rising edge is detected on the trigger signal

### Trigger signal

If send mode **On change** or **On trigger** is selected: This signal determines when a new HTTP(S) request is sent.

## Request settings

The request can be defined using the **Request settings** user interface.

The screenshot shows the 'Request settings' tab for an HTTP(S) request. The 'Method' is set to 'Get'. The 'Endpoint' field is empty. Below the endpoint field, there are sections for 'Path placeholders' and 'Query parameters', each with a table for defining placeholders and their corresponding test values.

## Endpoint

The URL can be defined using the **Endpoint** text box. The HTTP(S) module supports all standard HTTP methods which can be selected in the **Method** combo box.

This close-up shows the 'Method' dropdown menu set to 'Get' and the 'Endpoint' text box, which is highlighted with a red rectangle to indicate where the user should enter the URL.

An endpoint consists of the **path** and the **query**.

This close-up shows the 'Endpoint' text box filled with the example URL: 'https://httpstat.us/200?api-key=10ffbccdd&refresh=false'. The entire text box is highlighted with a red rectangle.

**Example**      *GET https://httpstat.us/200?api-key=10ffbccdd&refresh=false*  
**Method:**      GET  
**Path:**        https://httpstat.us/200  
**Query:**       api-key=10ffbccdd&refresh=false

## Query parameters and placeholders

Placeholders are used to make certain parts of the request dynamic. In those cases, instead of sending a static value, the current value of the defined signal is used.

For convenience, the query parameters are displayed in the query parameters grid. A query parameter always consists of a key and a value.

When pressing <RETURN> in the endpoint textbox or focusing a different control, the query parameters grid is updated. A new query parameter can also be added using the plus button next to the grid.

Following the example of <https://httpstat.us/200?api-key=10ffbcbccdd&refresh=false>, the query parameters grid contains two key-value pairs with four columns.

Query parameters			
Key	Test key	Value	Test value
✓ api-key		✓ 10ffbcbccdd	
✓ refresh		✓ false	

### Key

Defines the key of the query parameter. Can be a static value or a signal.

### Test key

If a signal is selected, the value in the test key column is used as the key for the query parameter when testing the request. Otherwise, the column is disabled.

### Value

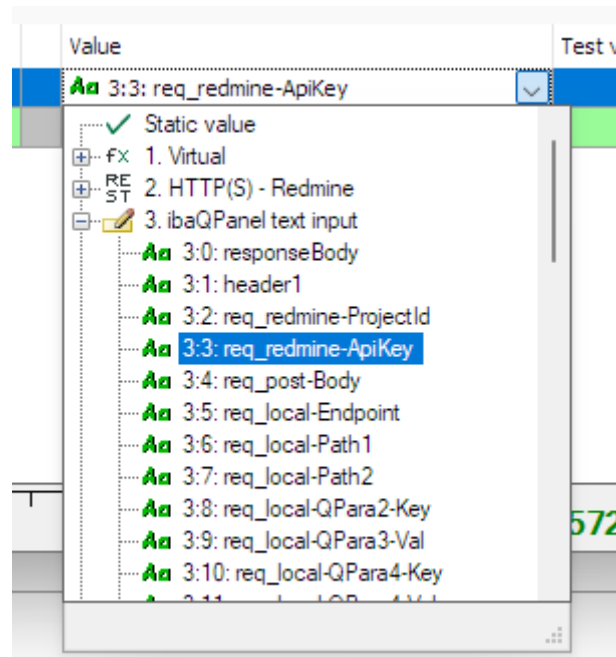
Defines the value of the query parameter. Can be a static value or a signal.

### Test value

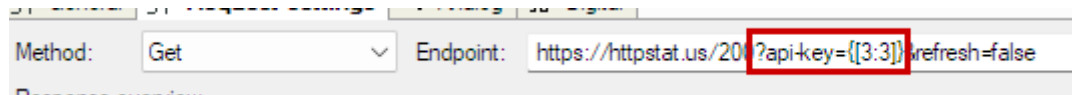
If a signal is selected, the value in the test value column is used as the value for the query parameter when testing the request. Otherwise, the column is disabled.

If the values in the cells are changed, the endpoint text box is updated automatically and vice versa.

In our example, the signal **[3:3]** is selected as the query parameter value.



After selecting the signal, the change is also reflected in the endpoint.



The full endpoint of the current configuration now looks like this:

<https://httpstat.us/200?api-key={{3:3}}&refresh=false>

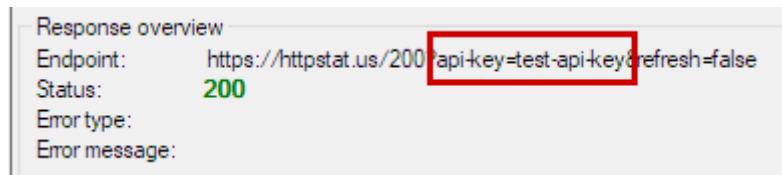
If the request is sent while the acquisition is running, the placeholder `{{3:3}}` will be replaced with the current value of the signal `[3:3]`.

For example, assuming signal `[3:3]` currently has the value `test`, the endpoint will be resolved to <https://httpstat.us/200?api-key=test&refresh=false>.

To test the request, a test value has to be entered for the value of the first query parameter. In this example, the value `test-api-key` is used.

Query parameters			
Key	Test key	Value	Test value
✓ api-key		Aa 3:3: req_redmine-ApiKey	test-api-key
✓ refresh		✓ false	

After using the **Test** button, the **Response overview** shows the endpoint successfully resolved the first query parameter to `api-key=test-api-key`.



## Path placeholders

The path placeholders grid is located above the query parameters grid and is used to dynamically change the path.

HTTP(S) (18) DEMO

General

Request settings

Analog

Digital

Method: Get

Endpoint: https://httpstat.us/200?api-key={{3:3}}&refresh=false

Test

Response overview

Endpoint: https://httpstat.us/200?api-key=test-api-key&refresh=false

Status: 200

Error type:

Error message:

Request

Response

Endpoint

Security

Headers

Body

Path placeholders

Placeholder	Signal	Test value

Query parameters

Key	Test key	Value	Test value
✓ api-key		Aa 3:3: req_redmine-ApiKey	test-api-key
✓ refresh		✓ false	



Contrary to the query parameters grid, the path placeholders grid only allows signals. Static values have to be defined directly in the endpoint textbox.

Path placeholders are used to dynamically change a part of the path.

The grid contains three columns.

### Placeholder

Defines a unique token that is used to identify this placeholder. The token is generated automatically but can be changed if needed.

### Signal

The signal which contains the value that will replace the placeholder.

### Test value

A test value that is used when testing the request.

To add a path placeholder, use the plus button next to the grid.

In this example the token is set to `{1}`, the signal `[3:6]` is selected and `200` defines the test value. Following the example, the endpoint textbox was updated automatically. The “200” of the original endpoint was removed manually since it’s replaced by the placeholder.

Method:	Get	Endpoint:	https://httpstat.us/{1}?api-key=[3:3]&refresh=false
---------	-----	-----------	---

When sending the request, `{1}` will be replaced by the test value `200`.

Response overview	
Endpoint:	https://httpstat.us/200?api-key=test-api-key&refresh=false
Status:	200
Error type:	
Error message:	

For convenience, instead of `{1}`, a more detailed token can be used (e.g. `{statusCode}`). This allows the user to understand the meaning of the placeholder more easily.

Path placeholders		
Placeholder	Signal	Test value
{statusCode}	[3:5] req_local-Endpoint	200

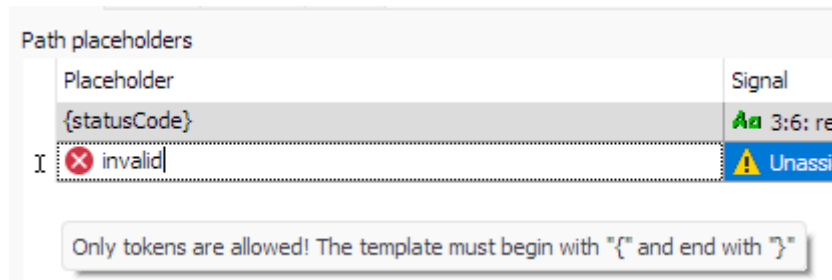
Endpoint:	https://httpstat.us/{statusCode}?api-key=[3:3]&refresh=false
-----------	--

### Remark:

Tokens must be unique and must start and end with a curly bracket.

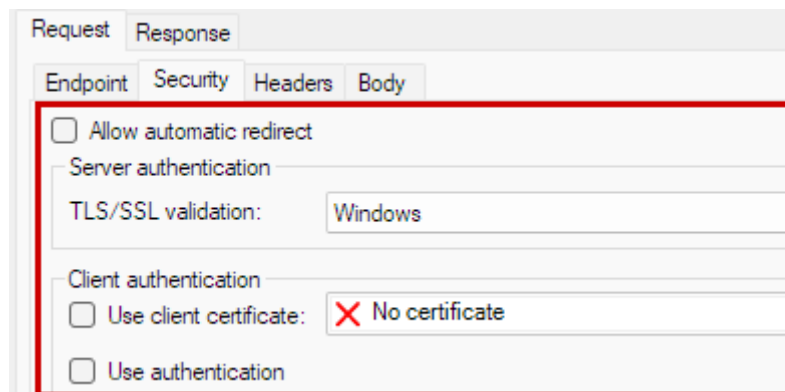
Path placeholders		
Placeholder	Signal	Test
{statusCode}	[3:6] req_local-Path1	200
{statusCode}	Unassigned	

⚠ Duplicate tokens are not allowed! Choose a different, unique token



## Security

The security tab provides various security related settings.

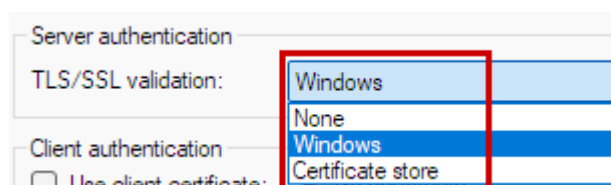


### Allow automatic redirect

If the destination responds with a redirect (HTTP status code 300 to 399), the redirection is followed if this option is activated.

### Server authentication

Defines the server SSL certificate validation behavior.



- **None:** The SSL certificate is not validated. This allows unsafe connections and is not recommended.
- **Windows:** The SSL certificate is validated automatically by the operating system.
- **Certificate store:** The SSL certificate is validated by the ibaPDA certificate store.

### Use client certificate

If enabled, sends the selected certificate as the client certificate as part of the request.

### Use authentication

If enabled, sends the request with the configured authentication.

## Authentication

ibaPDA currently supports three authentication methods.

- Basic Authentication
- Json Web Token
- OAuth2.0 (ROPC and client credentials)

### Basic authentication

The basic authentication allows choosing a username and a password. The authentication is added as the Authorization header to the request.

The screenshot shows two panels. The left panel, titled 'Settings', contains three fields: 'Type:' with a dropdown menu set to 'Basic', 'Add to:' with a dropdown menu set to 'Header', and 'Query parameter:' with an empty text box. The right panel, titled 'Basic', contains two fields: 'Username:' with a text box containing 'testUser' and 'Password:' with a text box containing seven asterisks.

### JSON Web Token (JWT)

The JSON Web Token can either be sent in the Authorization header as a bearer token, or as a query parameter in the URL with the provided key.

The first screenshot shows the 'Settings' panel for JWT. It has 'Type:' set to 'Json Web Token (JWT)', 'Add to:' set to 'Header', and an empty 'Query parameter:' field. The second screenshot shows the 'Settings' panel for JWT with 'Type:' set to 'Json Web Token (JWT)', 'Add to:' set to 'Query', and 'Query parameter:' set to 'my-auth-key'.

The screenshot shows the 'Json Web Token (JWT)' configuration panel. It has a title bar 'Json Web Token (JWT)'. Below it are three fields: 'Algorithm:' with a dropdown menu set to 'HS256', 'Secret:' with an empty text box, and 'Payload' with a large empty text area and a vertical scrollbar on the right.

**Algorithm**

Used for generating the signature. Currently only HMAC SHA256 is supported.

**Secret**

The secret is used as the key for the HMAC SHA256 algorithm.

**Payload**

The payload contains custom information used to validate the request.

**OAuth2.0**

The OAuth2.0 protocol defines multiple authorization standards. ibaPDA supports the ROPC (username/password) and the client credentials grant.

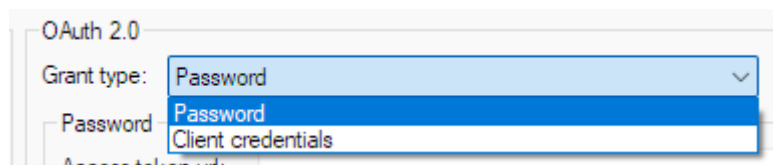
“Three-legged” grants are currently not supported.

**Remarks:**

Requests are automatically authorized using the given information. Subsequent requests will use the issued token.

If the token is expired and a refresh token is available, the token will be refreshed with the provided information.

To select a grant, choose one of the available options in the grant type combo box.



Some common properties are available for all grant types.

**Access token URL**

Defines the endpoint for getting and refreshing the token.

**Scope**

Optionally defines the requested scope of the token.

**Client Id**

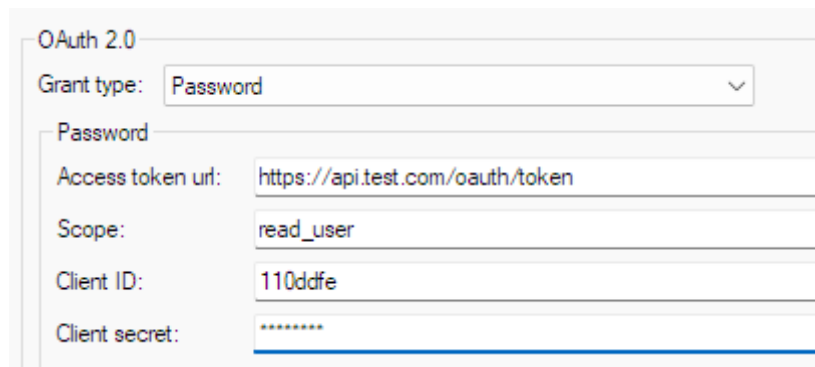
Optionally defines the client id.

**Client secret**

Optionally defines the client secret.

**Send client authentication in**

Defines whether the authentication for token-related requests is added to the header or the body



OAuth 2.0

Grant type: Password

Password

Access token url:

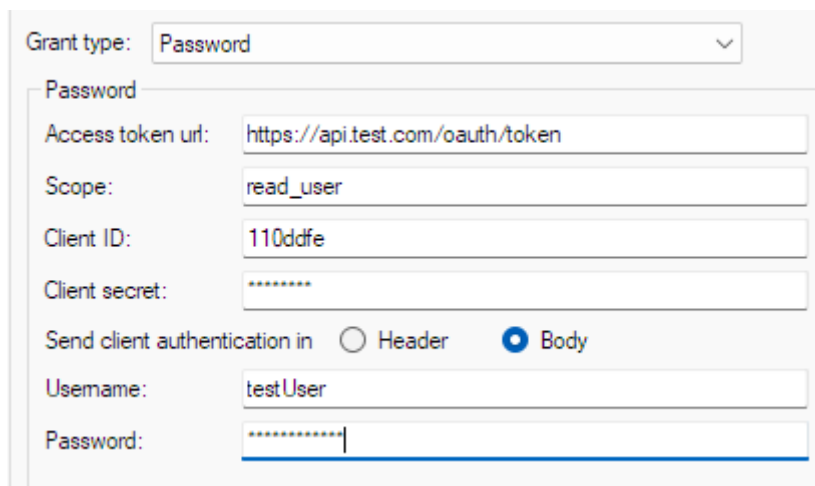
Scope:

Client ID:

Client secret:

### Password grant

Besides the common properties, the password grant type supports adding a username and a password.



Grant type: Password

Password

Access token url:

Scope:

Client ID:

Client secret:

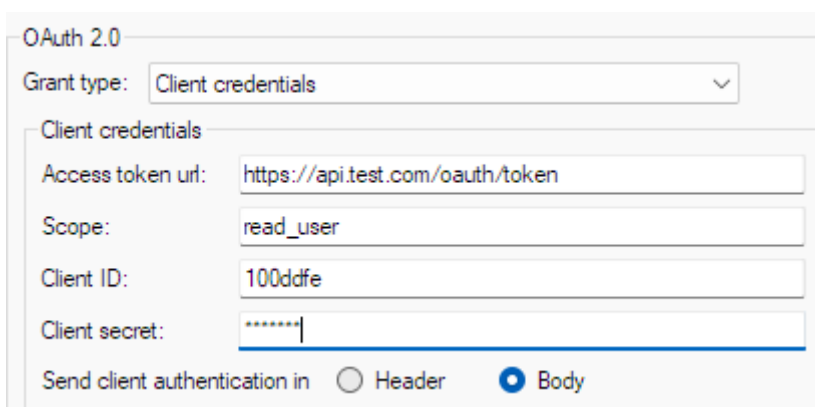
Send client authentication in ☐ Header ☒ Body

Username:

Password:

### Client credentials grant

This grant does not provide any other properties besides the common ones.



OAuth 2.0

Grant type: Client credentials

Client credentials

Access token url:

Scope:

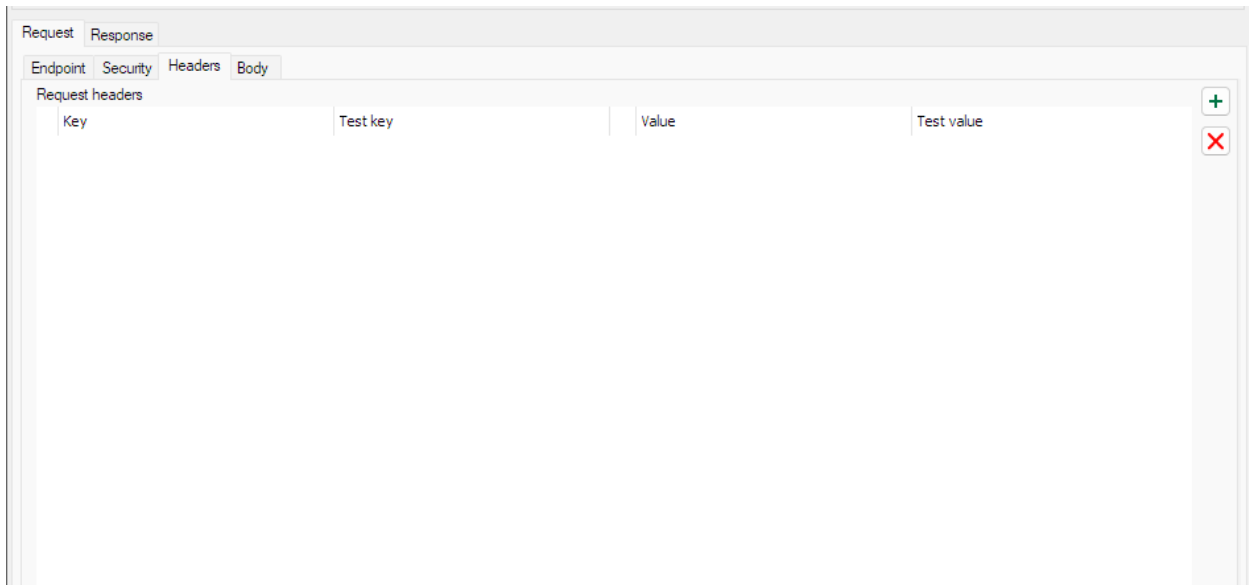
Client ID:

Client secret:

Send client authentication in ☐ Header ☒ Body

## Request headers

To define request headers, the request headers grid can be used.

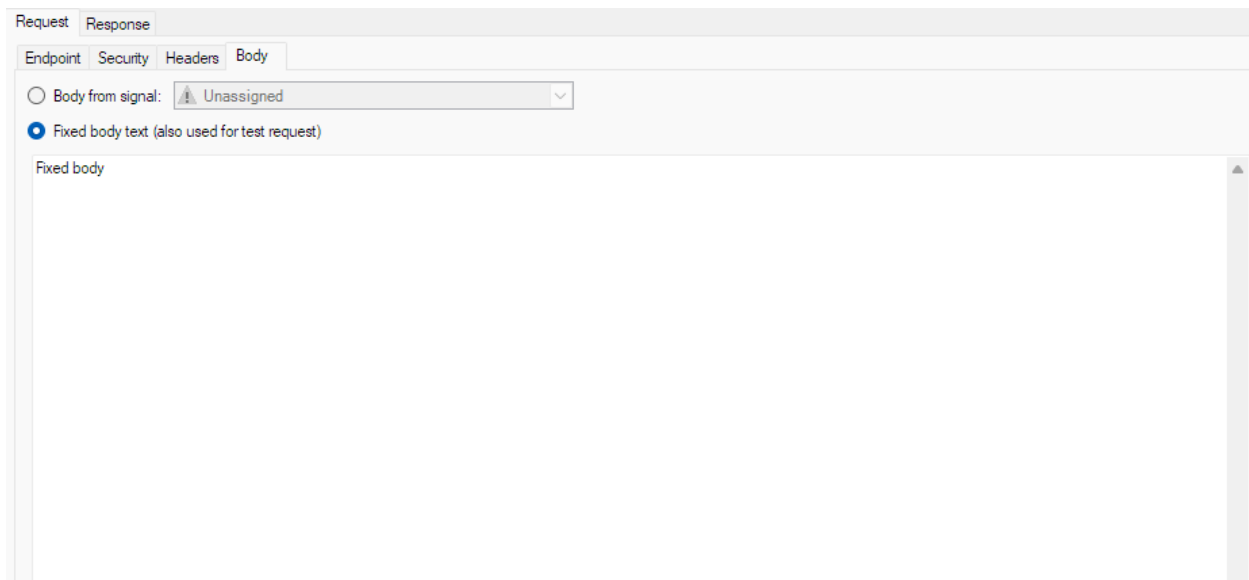


The screenshot shows the 'Request' tab in the ibaPDA interface. Under the 'Request' tab, there are sub-tabs: 'Endpoint', 'Security', 'Headers', and 'Body'. The 'Headers' sub-tab is selected. Below it, there is a 'Request headers' section. This section contains a table with two columns: 'Key' and 'Value'. The 'Key' column has a 'Test key' entry, and the 'Value' column has a 'Test value' entry. To the right of the table, there are two buttons: a green '+' button to add a new header and a red 'X' button to delete the selected header.

Like the query parameters, a static value or a signal can be selected for the key and the value. Use the plus button to add a new header and the delete button to delete the selected header.

## Request body

The request body can either be a fixed text or a signal.



The screenshot shows the 'Request' tab in the ibaPDA interface. Under the 'Request' tab, there are sub-tabs: 'Endpoint', 'Security', 'Headers', and 'Body'. The 'Body' sub-tab is selected. Below it, there is a 'Request body' section. This section contains two radio buttons: 'Body from signal: Unassigned' and 'Fixed body text (also used for test request)'. The 'Fixed body text' option is selected. Below the radio buttons, there is a text area labeled 'Fixed body'.

### Remark:

The fixed body textbox stays active if *Body from signal* is selected.

That's because the fixed body textbox can be used as the test value for testing the request.

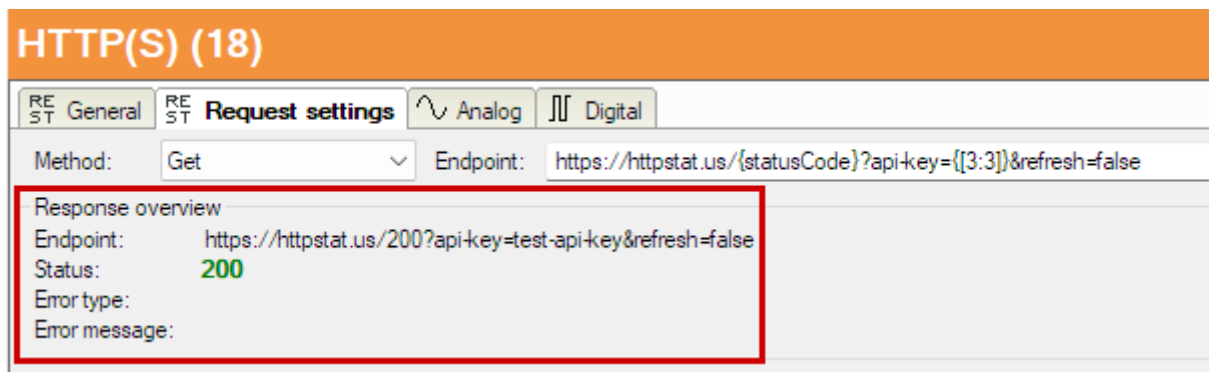
## Response settings

Response settings are related to the response received when sending the request.

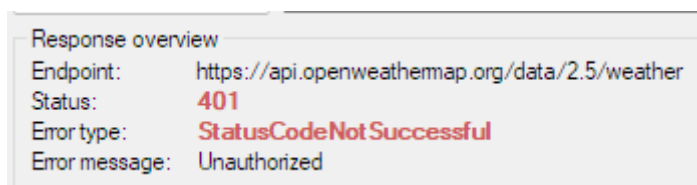
## Response overview

When testing a request, the response overview can be used as a source of quick information about the response.

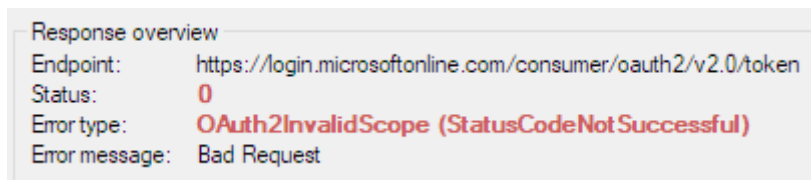
It contains the resolved endpoint, status code, error type and error message.



If the request failed, the error type and error message can be used to further identify the error.



Depending on the selected authentication, the response overview also shows errors that occurred during the authentication process.



## Error types

The following errors might occur.

Error type	Reason
RequestFailed	The request failed. No further information available
SslVerificationFailed	An error occurred while establishing an SSL connection
StatusCodeNotSuccessful	The HTTP status code is not successful (2xx)
Timeout	The server did not respond in time
InvalidEndpoint	The URL is not well formatted
IllegalPlaceholderCharacter	The value of a placeholder contains illegal characters

Error type	Reason
OAuth2MissingAuthorizationEndpoint	<b>Only applies to OAuth2</b> The access token URL is missing
OAuth2InvalidAuthorizationResponse	<b>Only applies to OAuth2</b> The request sent to the authorization server was not successful
OAuth2PasswordUsernameMissing	<b>Only applies to OAuth2: ROPC</b> The username or password is missing
OAuth2InvalidRequest	<b>Only applies to OAuth2</b> The request is invalid.  Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("invalid_request")
OAuth2InvalidClient	<b>Only applies to OAuth2</b> The client authentication failed.  Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("invalid_client")
OAuth2InvalidGrant	<b>Only applies to OAuth2</b> The authorization grant is invalid.  Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("invalid_grant")
OAuth2UnauthorizedClient	<b>Only applies to OAuth2</b> The client is not authorized for requesting the grant.  Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("unauthorized_client")
OAuth2UnsupportedGrantType	<b>Only applies to OAuth2</b> The requested grant is not supported  Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("unsupported_grant")
OAuth2InvalidScope	<b>Only applies to OAuth2</b> The request scope is not valid.



Error type	Reason
	Refer to <a href="https://datatracker.ietf.org/doc/html/rfc6749#section-5.2">https://datatracker.ietf.org/doc/html/rfc6749#section-5.2</a> for more details ("invalid_scope")
InvalidClientCertificate	The selected client certificate is not valid.
ServerCertificateInvalidDate	The date of the SSL certificate is not in a valid range.
ServerCertificateNotTrusted	The SSL certificate exists in the certificate store but is not trusted.
ServerCertificateInvalid	The SSL certificate is invalid.
ServerCertificateChainNotTrusted	The chain of the SSL certificate is not trusted.
ServerCertificateChainNotValid	The chain of the SSL certificate is not valid.
ServerCertificateMissingInManager	The SSL certificate is missing in the certificate store.

## Response attributes

The response attributes contain information about the response, such as the duration, status code, error type, error message and endpoint.

Request

Response

General

Headers

Body

Response attributes

Attribute	Value
▶ Duration	00:00:00.1460000
Status Code	200
Error Type	None
Error Message	
Endpoint	https://api.openweathermap.org/data/2.5/weather?

The module also contains auto-generated signals for those attributes.

HTTP(S) - Weather (6)		
RE	General	Request settings
ST	Analog	Digital
Name	Unit	Active
0 Response status code		<input checked="" type="checkbox"/>
1 Response duration		<input checked="" type="checkbox"/>
2 Response error type		<input checked="" type="checkbox"/>
3 Response error message		<input checked="" type="checkbox"/>
4 Response body		<input checked="" type="checkbox"/>

## Response headers

The response headers grid contains the headers received in the response.

Request Response

General Headers Body

Add header signals Remove header signals

Key	Value
Access-Control-Allow-Credentials	true
Access-Control-Allow-Methods	GET, POST
Access-Control-Allow-Origin	*
Connection	keep-alive
Date	Tue, 16 Apr 2024 09:29:12 GMT
Server	openresty
X-Cache-Key	/data/2.5/weather?lang=de&lat=49.33&lon=12.18&units=metric

Using the **Add header signals** and the **Remove header signals** buttons, the headers can either be added to or removed from the signals of the module.

Request Response

General Headers Body

Add header signals Remove header signals

Response headers

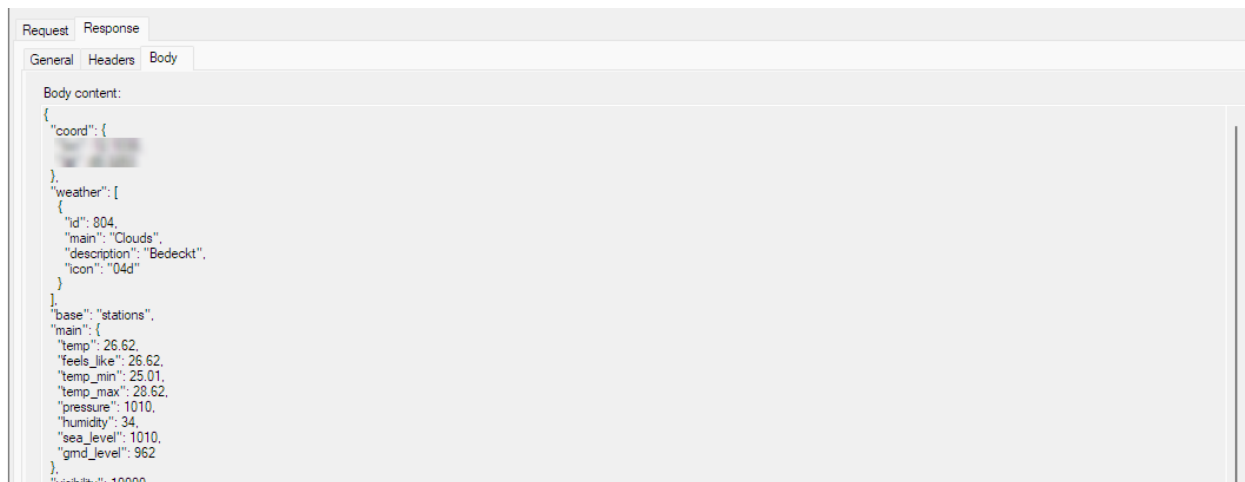
Key

## HTTP(S) - Weather (6)

RE ST	General	RE ST	Request settings	Analog	Digital
Name	Unit	Active			
0 Response status code		<input checked="" type="checkbox"/>			
1 Response duration		<input checked="" type="checkbox"/>			
2 Response error type		<input checked="" type="checkbox"/>			
3 Response error message		<input checked="" type="checkbox"/>			
4 Response body		<input checked="" type="checkbox"/>			
5 Connection		<input checked="" type="checkbox"/>			
6 Access-Control-Allow-Credentials		<input checked="" type="checkbox"/>			
7 Access-Control-Allow-Methods		<input checked="" type="checkbox"/>			
8 Access-Control-Allow-Origin		<input checked="" type="checkbox"/>			
9 Connection		<input checked="" type="checkbox"/>			
10 Date		<input checked="" type="checkbox"/>			
11 Server		<input checked="" type="checkbox"/>			
12 X-Cache-Key		<input checked="" type="checkbox"/>			

## Response body

The response body contains the received body.



The body is also available as an auto-generated signal in the analog tab.

HTTP(S) - Weather (6)

RE ST	General	RE ST	Request settings	Analog	Digital
	Name	Unit	Active		
0	Response status code		<input checked="" type="checkbox"/>		
1	Response duration		<input checked="" type="checkbox"/>		
2	Response error type		<input checked="" type="checkbox"/>		
3	Response error message		<input checked="" type="checkbox"/>		
4	Response body		<input checked="" type="checkbox"/>		
5	Connection		<input checked="" type="checkbox"/>		
6	Access-Control-Allow-Credentials		<input checked="" type="checkbox"/>		
7	Access-Control-Allow-Methods		<input checked="" type="checkbox"/>		
8	Access-Control-Allow-Origin		<input checked="" type="checkbox"/>		
9	Connection		<input checked="" type="checkbox"/>		
10	Date		<input checked="" type="checkbox"/>		
11	Server		<input checked="" type="checkbox"/>		
12	X-Cache-Key		<input checked="" type="checkbox"/>		

## Analog and digital signals

The HTTP(S) module provides auto-generated signals.

### Analog signals

By default, each HTTP(S) module contains these signals:

- Response status code
- Response duration
- Response error type
- Response error message
- Response body
- Endpoint

HTTP(S) - teams oauth (17)		
RE ST	General	RE ST Request settings
	Analog	Digital
Name	Unit	Active
0 Response status code		<input checked="" type="checkbox"/>
1 Response duration		<input checked="" type="checkbox"/>
2 Response error type		<input checked="" type="checkbox"/>
3 Response error message		<input checked="" type="checkbox"/>
4 Response body		<input checked="" type="checkbox"/>
5 Endpoint		<input checked="" type="checkbox"/>

The signals always contain the values of the last request.

### Digital signal

Each HTTP(S) module also provides the digital signal Pending request.

This signal indicates whether a request is currently active (1) or not (0).

HTTP(S) - teams oauth (17)		
RE ST	General	RE ST Request settings
	Analog	Digital
Name	Unit	Active
0 Pending request		

## Validation warnings

The following warnings and errors may occur during the validation.

### The endpoint cannot be empty

The endpoint must contain a value.

HTTP(S) - Weather (6)

RE ST General RE ST Request settings Analog Digital

Method: Get Endpoint:

### The endpoint must start with http, https or a placeholder

The endpoint is filled, but doesn't start with http, https or a placeholder.

HTTP(S) - Weather (6)

RE ST General RE ST Request settings Analog Digital

Method: Get Endpoint: iba-ag.com

To fix this, prefix the address with http, https or use a placeholder that is filled with either http or https during the acquisition.

http://iba-ag.com

https://iba-ag.com

{protocol}://iba-ag.com

### The placeholder {...} is not assigned to a signal

A placeholder is used but no signal is assigned.

HTTP(S) (18)

RE ST General RE ST Request settings Analog Digital

Method: Get Endpoint: https://httpstat.us/{statusCode}/api-key=([3:3])&refresh=false Test

Response overview

Endpoint:

Status:

Error type:

Error message:

Request Response

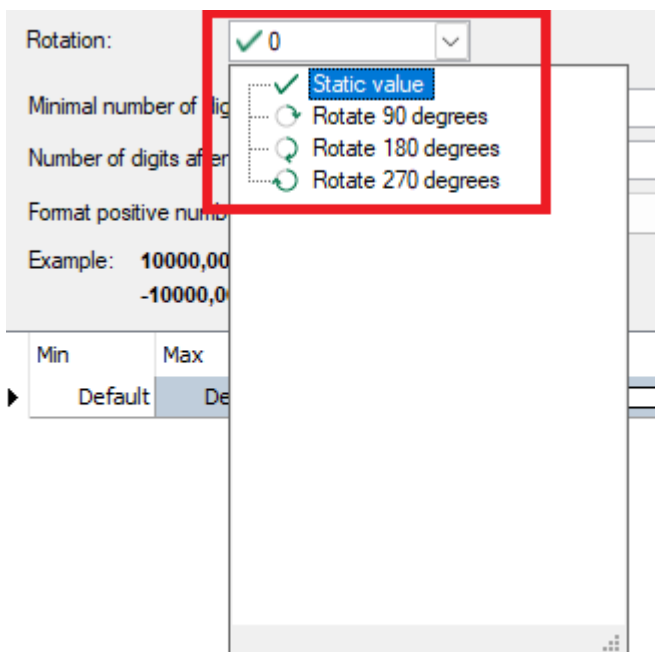
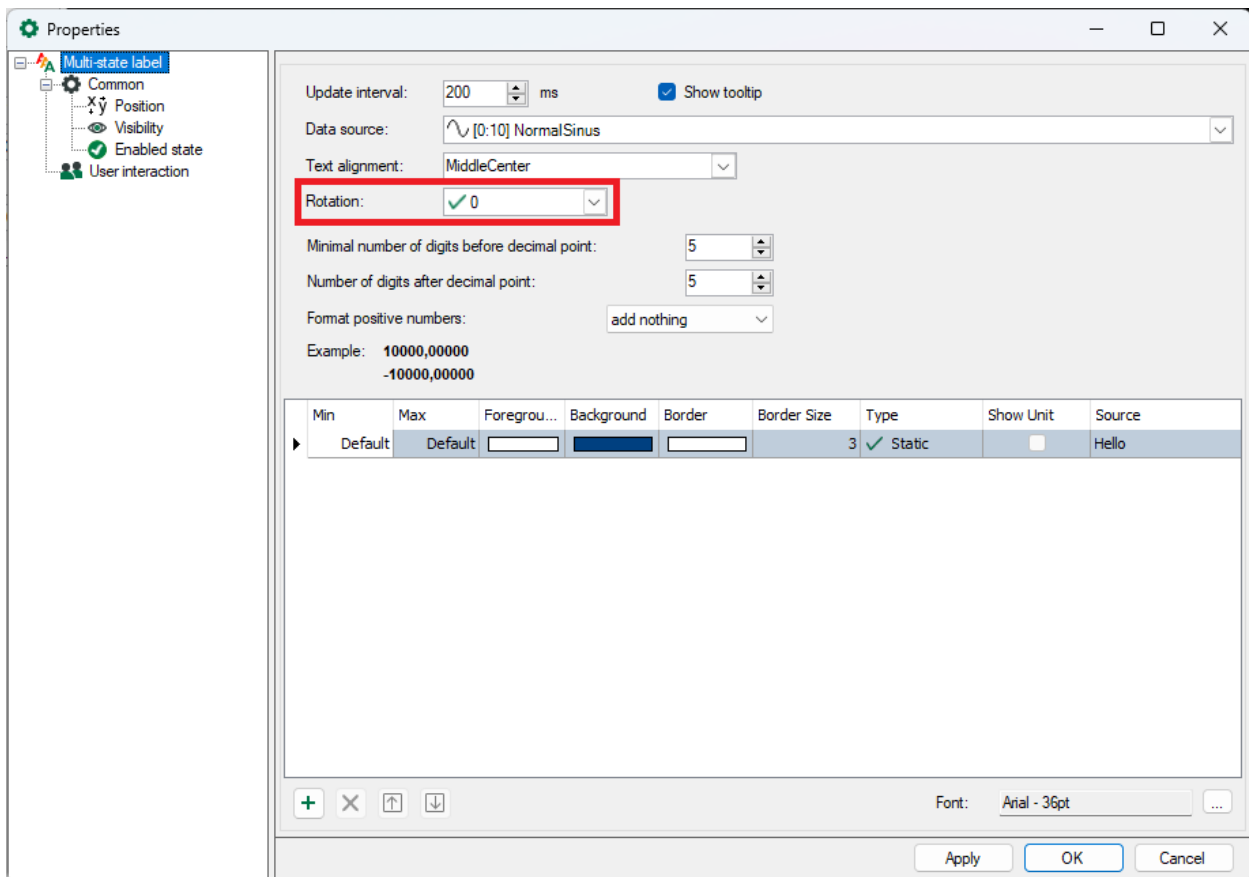
Endpoint Security Headers Body

Path placeholders

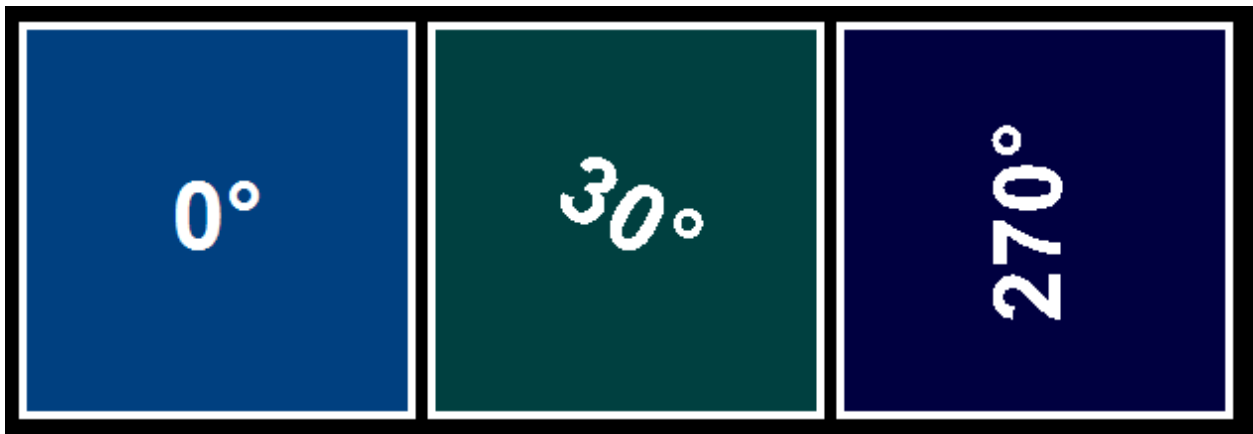
Placeholder	Signal	Test value
{statusCode}	Unassigned	200

## Multi-state label text rotation

The multi-state label view in ibaQPanel now has an option for text rotation just like the normal label view. The value determines how many degrees the text of the view is rotated. You can enter a custom value or choose a preset rotation of 90, 180 or 270 degrees.



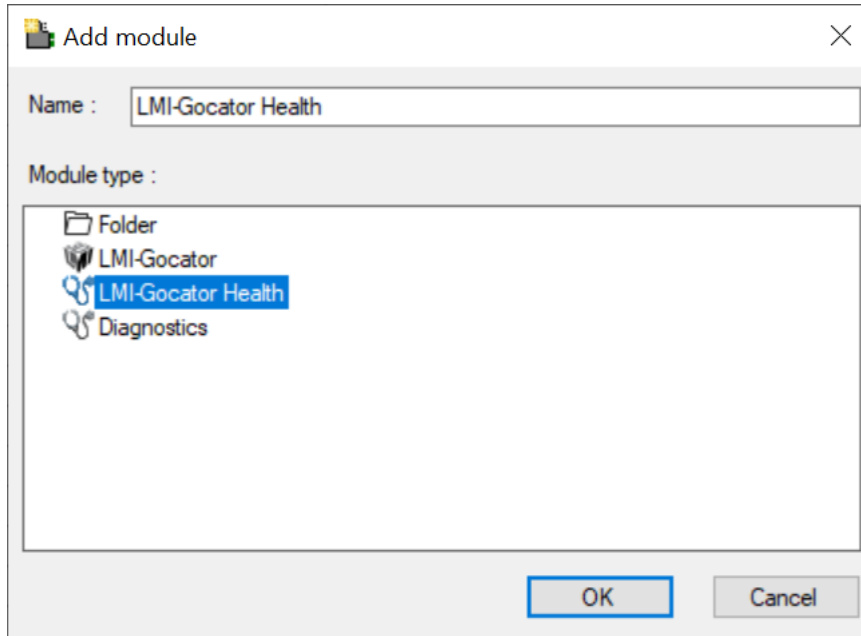
Three multi-state label views with text rotation of 0, 30 and 270 degrees respectively:



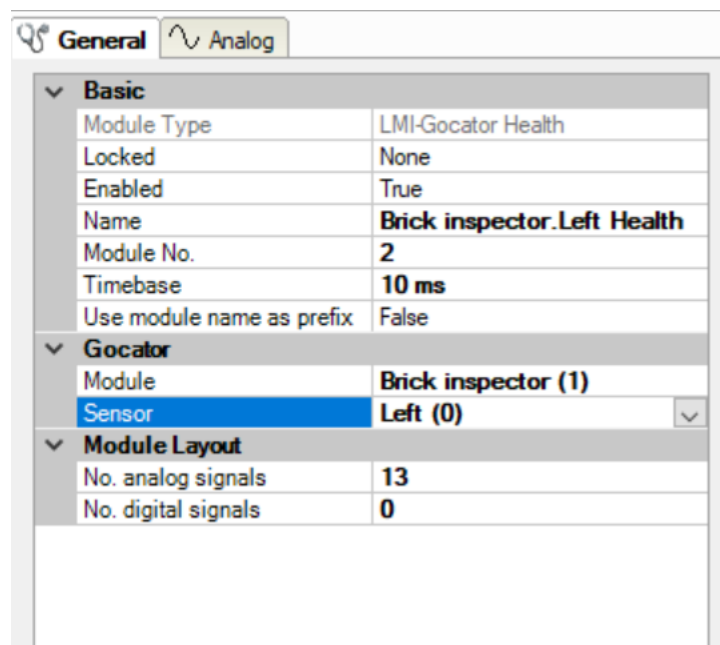
### LMI-Gocator Health module

When LMI Gocator sensors send measurement data to ibaPDA, every second they also send diagnostic information, e.g. sensor temperature, current speed, total number of scanned profiles, .... Using the new LMI-Gocator Health module it is now possible to measure this data.

To do so, add a new module under the LMI-Gocator interface node in the I/O manager.



Select **LMI-Gocator Health** and click **OK**.



A Gocator Health module must always be linked to a regular LMI-Gocator module (that measures profile data). The module to which the health module is linked can be selected using the **Module** property in the **Gocator** section. In case several Gocator sensors are configured in the LMI-Gocator module it is also required to select for which sensor the diagnostic or health data should be measured. This is done using the **Sensor** property.



Apart from those two module-specific properties, all standard module properties (e.g. Name, Locked, Enabled, ...) can be configured here as well.

When creating an LMI-Gocator Health module, a standard list of common signals is automatically created. It is, however, possible to add or remove signals by changing the **No. analog signals** and **No. digital signals** properties.

The actual module signals can be configured in the **Analog** and **Digital** tab.

Brick inspector.Left Health (2) DEMO

General Analog

Name	Unit	Gain	Offset	Type	DataType	Active	Actual
0 Current system state		1	0	STATE	DINT	<input checked="" type="checkbox"/>	1
1 The state of the sensor alignment		1	0	ALIGNMENT_STA...	DINT	<input checked="" type="checkbox"/>	0
2 The number of scanned profiles or surfaces		1	0	SCAN_COUNT	DINT	<input checked="" type="checkbox"/>	2400
3 Sensor synchronization source		1	0	SYNC_SOURCE	DINT	<input checked="" type="checkbox"/>	1
4 Internal temperature	°C	1	0	TEMPERATURE	FLOAT	<input checked="" type="checkbox"/>	35,937 °C
5 CPU temperature	°C	1	0	CPU_TEMPERAT...	FLOAT	<input checked="" type="checkbox"/>	42,389 °C
6 CPU usage	%	1	0	CPU_USED	DINT	<input checked="" type="checkbox"/>	9 %
7 System encoder tick		1	0	ENCODER_VALUE	DINT	<input checked="" type="checkbox"/>	1
8 System encoder frequency	ticks/s	1	0	ENCODER_FREQ...	DINT	<input checked="" type="checkbox"/>	0 ticks/s
9 Current speed	Hz	1	0	SPEED	DINT	<input checked="" type="checkbox"/>	104 Hz
10 Maximum speed	Hz	1	0	CPU_TEMPERATURE		<input checked="" type="checkbox"/>	212 Hz
11 Last reported processing latency value	µs	1	0	CAMERA_0_TEMPERATURE		<input checked="" type="checkbox"/>	6419 µs
12 Maximum reported processing latency	µs	1	0	CAMERA_1_TEMPERATURE		<input checked="" type="checkbox"/>	7827 µs

0 256 512 768 1024 1280 1536 1792 ∞ 1014 OK Apply Cancel

MEMORY\_USED  
MEMORY\_CAPACITY  
STORAGE\_USED  
STORAGE\_CAPACITY  
CPU\_USED  
SYNC\_SOURCE  
NET\_OUT\_USED  
NET\_OUT\_RATE  
NET\_OUT\_CAPACITY  
NET\_OUT\_LINK\_STATUS  
DIGITAL\_INPUTS  
EVENT\_COUNTS  
CAMERA\_SEARCH\_COUNT  
CAMERA\_TRIGGER\_DROPS  
STATE  
SPEED

Apart from the standard signal properties, the type of health signal must be configured here. ibaPDA contains a built-in list with all known health signal types at the time of implementation. Each health signal type corresponds to a unique integer value (as defined in the LMI Gocator SDK). Every second, a Gocator sensor sends a list of pairs where each pair contains an ID (= the integer value corresponding to the health signal type) and the actual diagnostic value. ibaPDA goes through this list and checks for each entry whether it should be measured or not.

If a health signal type is required that is not contained in ibaPDA's built-in list, it is possible to manually enter the integer value in the **Type** column of the signal grid. Make sure the **DataType** is also set accordingly (this is done automatically for known health signal types).

For debugging purposes or simply to obtain a good overview of ibaPDA's built-in health signal types, it is possible to automatically add all known health signals to a health module. To do so, select the health module in the I/O manager while holding the SHIFT key on your keyboard; if the module was already selected, first deselect it by selecting another node. In the **General** tab of the module, a link will appear at the bottom

**General** **Analog**

▼ **Basic**

Module Type	LMI-Gocator Health
Locked	None
Enabled	True
Name	<b>Brick inspector.Left Health</b>
Module No.	2
Timebase	10 ms
Use module name as prefix	False

▼ **Gocator**

Module	Brick inspector (1)
Sensor	Left (0) ▼

▼ **Module Layout**

No. analog signals	13
No. digital signals	0

**Sensor**  
The index of the sensor for which the health data should be measured.

[Add all known signals](#)

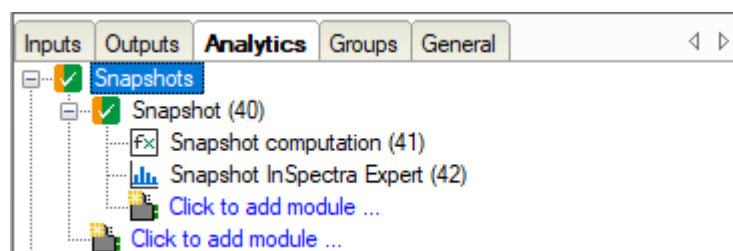
When pressing this link, all currently defined signals will be removed and a list of all known health signals will be added instead.

## Snapshots interface

The Snapshots interface can be used to buffer signal data for a period of time, after which calculations can be performed on the data, using a computation or InSpectra Expert module; this is how a “snapshot” is described in this context. Once the calculations have been done the results are written to a DAT file, along with the buffered signal data (if required). Note that the calculations and writing of the DAT files is executed in an asynchronous way, meaning that the recording of other signal data is not hindered by possibly long calculation times of snapshot modules.

Order Information	
Order no.	30.770026
Name	ibaPDA-Snapshot
License	

In case the Snapshots license is available, the **Snapshots** interface can be found under the **Analytics** tab in the I/O Manager



Under the **Snapshots** interface node **Snapshot** modules can be added. When adding such a module, the basic module properties (e.g. **Name**, **Enabled**, **Comment**, ...) are available as well as the possibility to select a **Profile**. A profile, as is the case for e.g. InSpectra modules, can be referenced by multiple modules and thus allows the user to easily change certain module properties for all modules simultaneously.

**General** Storage Analog Digital

**Basic**

Module Type	Snapshot
Locked	None
Enabled	True
Name	Snapshot
Comment	
Module No.	40
Timebase	10 ms
Use module name as prefix	False

**Profile**

Profile	SnapshotProfile
Condition 1	[2:1] KPI 1
Condition 2	[2:2] KPI 2
Condition 3	[2:3] KPI 3

Custom static value

- 1. ibaMS8xIEPE
- 2. Slow signals
  - 2:0: Speed
  - 2:1: KPI 1
  - 2:2: KPI 2
  - 2:3: KPI 3
- 3. ibaQPanel input

[Configure profiles](#)

An existing profile can be selected or a new profile can be created, either by clicking the link **Configure profiles** at the bottom of the module's **General** tab or by opening the drop-down menu next to the **Profile** property. A new dialog will appear listing all known snapshot profiles.

**Configure profiles**

Profiles

- SnapshotProfile

**General** Conditions

**General**

Interval	120 min
Snapshot size	300 s
Number of sections	10
Condition validation	Snapshot
Max snapshots per interval	1

**Interval**

Defines the interval in minutes in which "max snapshots per interval" are taken. If the max number of snapshots is reached, the module will wait for the start of the next interval before taking new snapshots.

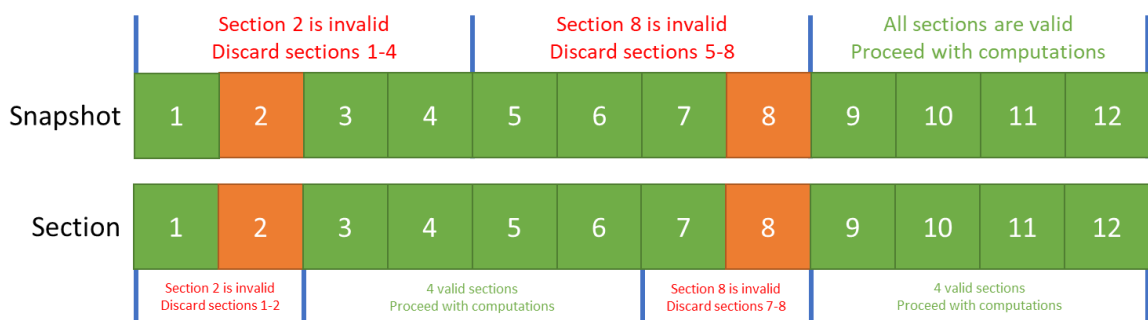
OK Cancel

A snapshot profile consists of a **General** and a **Conditions** section. The **Conditions** section allows the user to only allow generation of snapshots in case a certain set of conditions are met (based on signal values).

### General properties

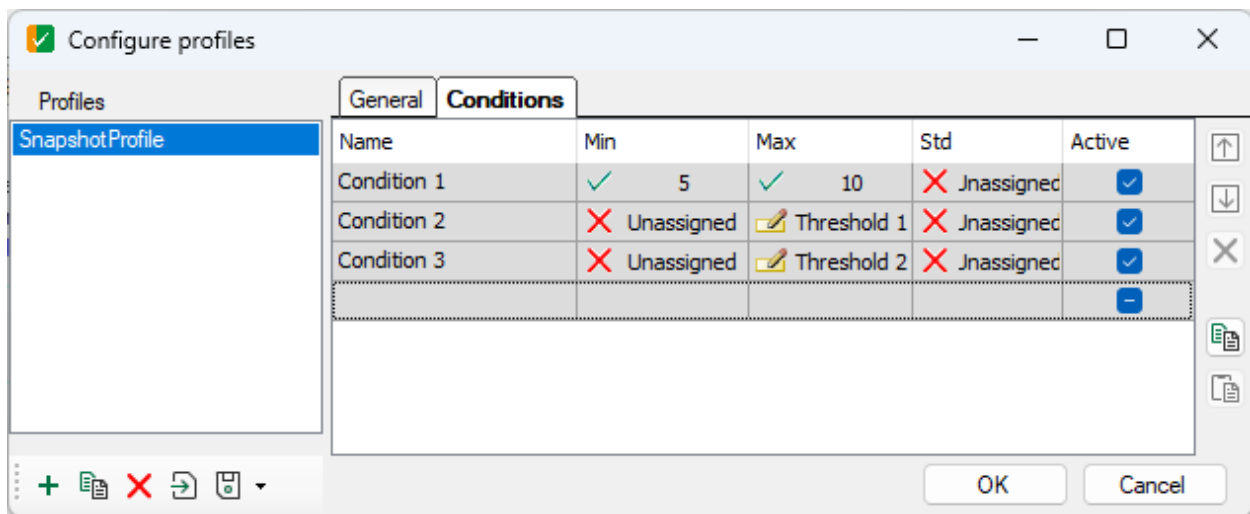
- **Interval:** the duration of one cycle in which the **maximum number of snapshots per interval** parameter is evaluated. If the defined maximum number of snapshots per interval is reached, the module will wait for the start of the next interval before producing new snapshots
- **Snapshot size:** the duration of one snapshot (i.e. the length in seconds of the buffered signal data)
- **Number of sections:** a snapshot can be divided in a number of isochronous sections where the conditions are then checked for each section individually
- **Condition validation:** determines how the conditions to generate a snapshot should be checked. The options **Snapshot** or **Section** are available:
  - **Snapshot:** as soon as one of the validation sections doesn't meet the conditions the entire snapshot is regarded as invalid. For example: if a snapshot is divided into 8 sections and the second section fails to meet the validation criteria, ibaPDA will ignore data for the following 6 sections. Only then can ibaPDA start a new snapshot.
  - **Section:** as soon as one of the validation sections doesn't meet the conditions, the buffered data for that section and previously buffered sections is regarded as invalid and will be ignored. A new snapshot can be started at the start of the next section.

The difference between **Snapshot** and **Section** mode is further illustrated in this picture (where the number of sections per snapshot is 4):



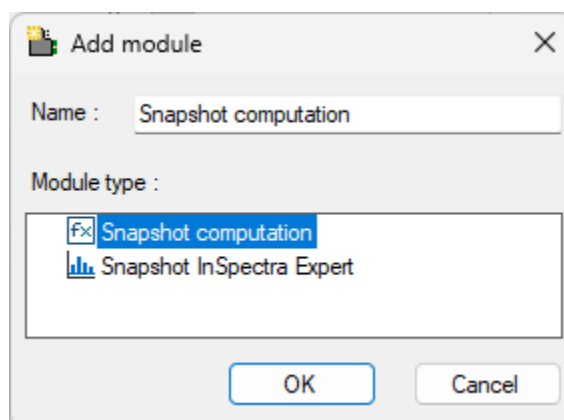
- **Max snapshots per interval:** once the maximum number of snapshots in an interval has been reached, the snapshot module will stop producing snapshots (even if the conditions are met) and wait until the next snapshot interval before creating new snapshots

## Conditions



In the **Conditions** tab a table is shown where the user can define a number of conditions. A condition basically is a signal for which the minimum (**Min**) and maximum (**Max**) value and the standard deviation (**Std**) can be checked. A static value or signal can be defined for these 3 properties. In case a property should not be checked, it should be set to **Unassigned**. Note that the signals are not directly selected here but instead, placeholders are generated (with the **Name** configured in the table here) to which signals can be assigned in the snapshot module's **General** tab.

A snapshot module on its own merely buffers data in case certain conditions are met. What should be done with this data (i.e. calculations) is defined in the submodules of a snapshot module.



Two submodule types are available: the **Snapshot computation** module and the **Snapshot InSpectra Expert** module. Both modules function in more or less the same way as their non-snapshot counterparts where the main difference lies in the fact that these modules operate on buffered data and the original modules operate on live data.

The Snapshot computation module has one additional feature: **Indicators**. These are single values that are evaluated over the entire snapshot. **Indicators** can be defined when configuring the profile of a snapshot computation module:

	Name	Expression	Unit	Intermediate
0	Eff_speed	$f_x$ EffInTime({Speed},{slen})	?	<input type="checkbox"/>
1	Median_speed	$f_x$ MedianInTime({Speed},{slen})	?	<input type="checkbox"/>
2	Max_speed	$f_x$ Max({Speed})	?	<input type="checkbox"/>
		$f_x$	?	<input checked="" type="checkbox"/>

As is the case for **Analog** and **Digital** signals (which can also be found in the regular computation module), **Placeholders** can be defined and used in the expressions. An additional placeholder **{slen}** is automatically defined which is translated into the snapshot duration [s].

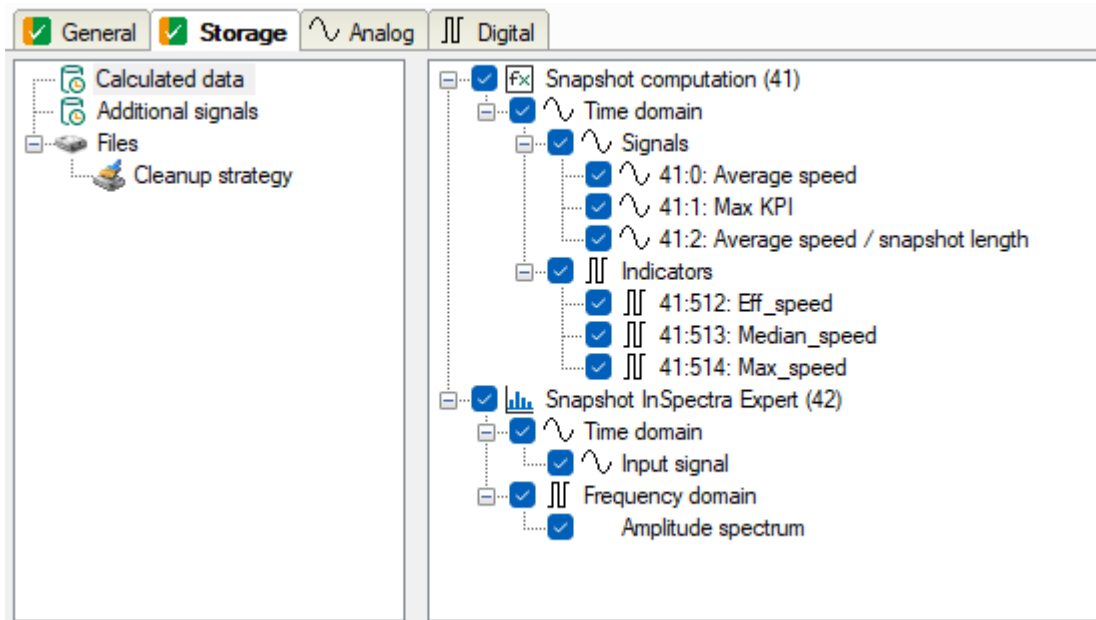
The **Analog** tab of a snapshot computation module consists of two sections (each of which is only displayed in case it actually contains signals): **Signals** and **Indicators**.

	Name	Expression	Unit	Act...	Actual
<b>Signals</b>					
0	Average speed	Avg([2:0])		<input checked="" type="checkbox"/>	
1	Max KPI	Max([2:2])		<input checked="" type="checkbox"/>	
<b>Indicators</b>					
512	Eff_speed	EffInTime([2:0},{slen})		<input checked="" type="checkbox"/>	
513	Median_speed	MedianInTime([2:0},{slen})		<input checked="" type="checkbox"/>	
514	Max_speed	Max([2:0])		<input checked="" type="checkbox"/>	

Indicator signals always start at signal number 512, meaning that the number of regular signals is limited to 512. Note that indicator signals are updated only once per snapshot (since they represent a single value).

A Snapshot InSpectra Expert is configured in the same way as a regular InSpectra Expert module. Only the **Snapshots** section, which refers to the built-in snapshot functionality of the InSpectra interface and NOT the Snapshots interface, is unavailable for the Snapshot InSpectra Expert module.

Once a snapshot module has finished creating a snapshot (including the calculations defined by its submodules), the results will be written to disk. This step can be configured in the **Storage** tab of the snapshot module.



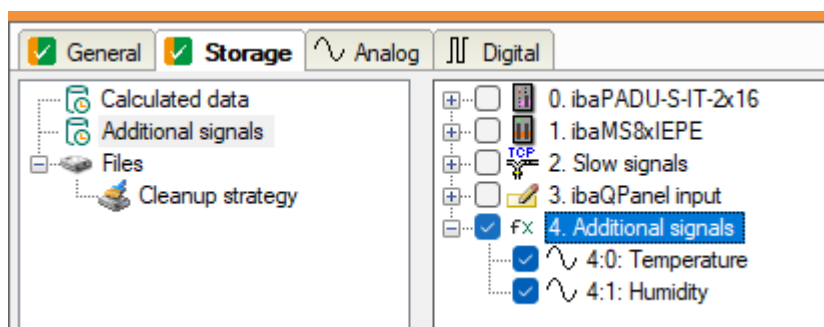
When selecting the **Calculated data** node all signals provided by the snapshot's submodules are displayed in the right panel. Per submodule a **Time domain** and **Frequency domain** group are displayed (if applicable) where each group can have a **Signals** and **Indicators** subgroup. The reason for this is that for each snapshot, the snapshot creates up to 4 files:

- Time domain data: a DAT file with the suffix `_trs` containing all time domain signals
- Spectra: a DAT file with the suffix `_fft` containing all frequency domain data
- Envelope spectra: a DAT file with the suffix `_fht` containing all frequency domain data from InSpectra modules that calculate an envelope spectrum
- Indicator data: an XML file with the suffix `_trd` containing all calculated indicator values

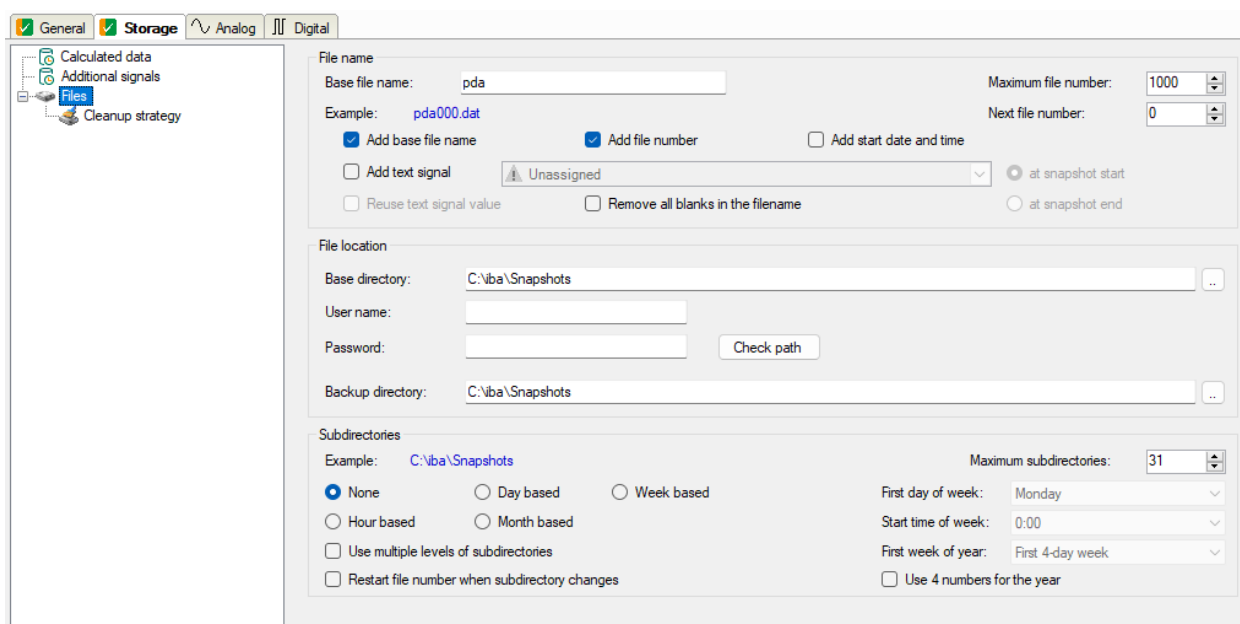
In the right panel it is possible to select signals that should be included in the generated snapshot files.

When selecting the **Additional signals** node, a signal tree (excluding snapshot modules and their submodules) is displayed in the right panel. Here, the user can select signals that should be buffered by the snapshot module and written to the time domain snapshot file.

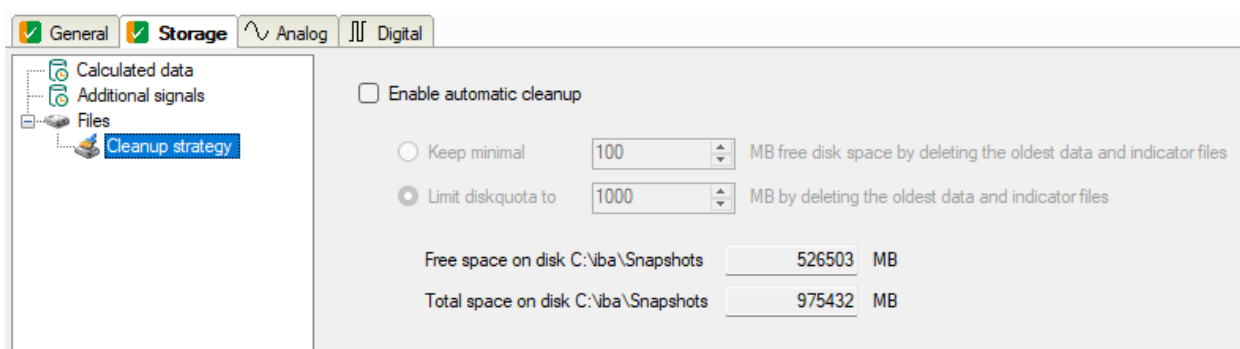




The **Files** node allows the user to configure the format of the snapshot filenames and the location to which the files should be written. This dialog is very similar to the one used to configure a time-based data store.



In the **Cleanup strategy** subnode, a cleanup strategy can be defined, again very similar to the cleanup strategy of a time-based data store. The option to keep a maximum or to keep at least a number of files is not available in this case.



A snapshot module also contains a small set of diagnostic signals which are made available in the **Analog** and **Digital** tabs.

<div> <div>General</div> <div>Storage</div> <div>Analog</div> <div>Digital</div> </div>			
Name	Unit	Active	Actual
0 Generated snapshots		<input checked="" type="checkbox"/>	
1 Failed snapshots		<input checked="" type="checkbox"/>	
2 Processing time	ms	<input checked="" type="checkbox"/>	

The **Analog** tab contains the following signals:

- **Generated snapshots:** number of times a snapshot could be successfully generated
- **Failed snapshots:** number of times generation of a snapshot was impeded because of a section that didn't meet the defined conditions
- **Processing time:** the amount of processing time it took for the latest snapshot to calculate all signals and/or indicators defined in the snapshot's submodules. This does not include the time required to write the resulting files to disk.

<div> <div>General</div> <div>Storage</div> <div>Analog</div> <div>Digital</div> </div>			
Name	Acti...	Actual	
0 Buffering	<input checked="" type="checkbox"/>		

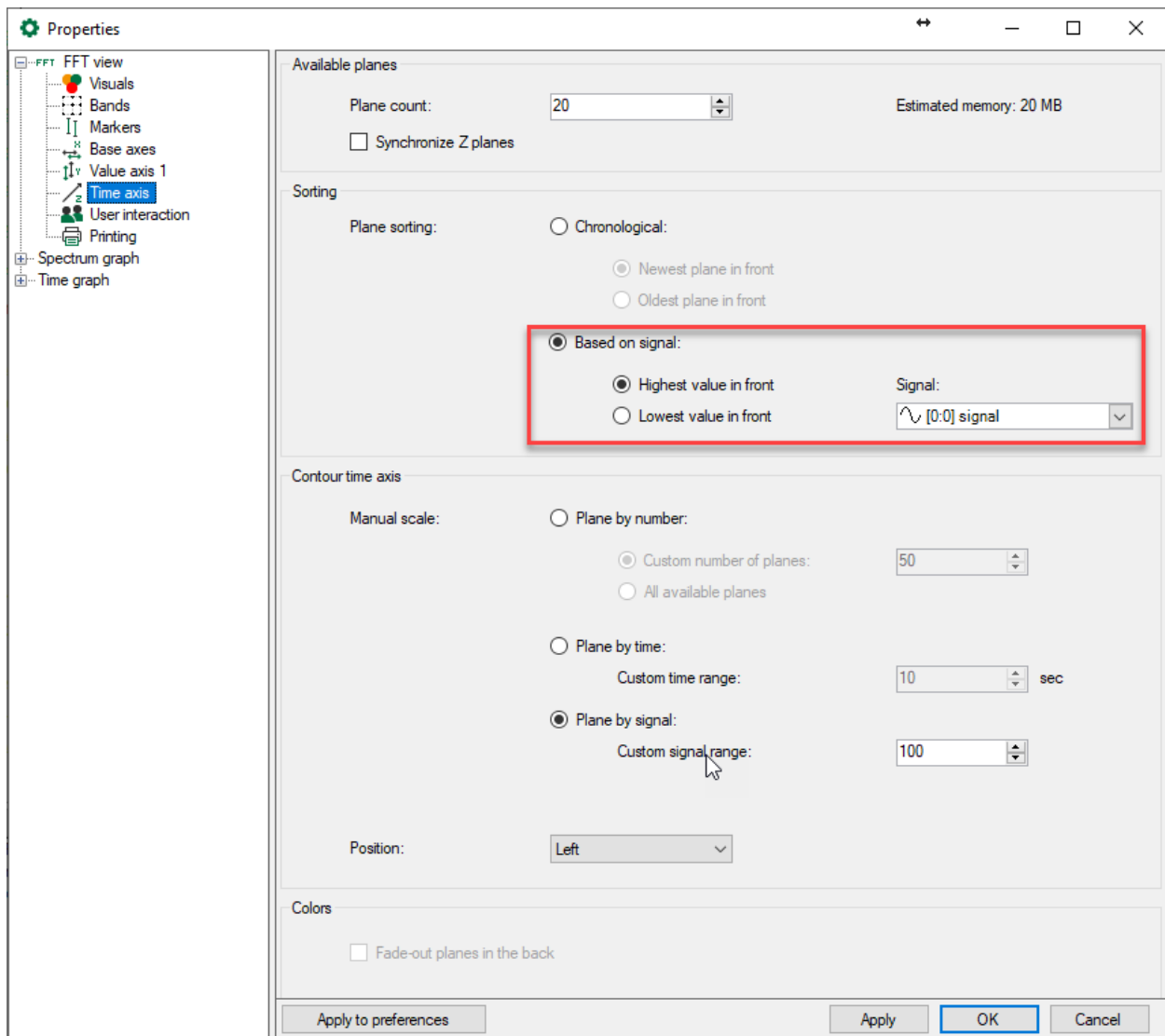
The **Digital** tab only contains the **Buffering** signal which is asserted whenever the snapshot module is in the process of buffering data (while checking whether snapshot conditions are met).

## FFT view and Cycle view

### Plane sorting based on the value of a process signal

In previous versions, the planes in the waterfall are sorted by the timestamp of the FFT/Cycle. Two options are available “Newest plane in front” and “Oldest plane in front”.

In this version, it is also possible to configure sorting based on the value of a custom process signal. For each FFT/Cycle, an average value of this process signal is calculated. Two options are available for the sorting: “Highest value in front” or “Lowest value in front”. The sorting happens on the fly: Each time a new FFT/Cycle comes in, all the planes are sorted again. This applies to the contour plot and the waterfall.



### Scaling of the contour Z-axis based on the value of a process signal

In previous versions, the distance between the planes in the contour plot was either fixed (“Plane by number”) or depending on the time difference between planes (“Plane by time”).

In this version, we introduce a third option “Plane by signal”. This option can only be used if the plane sorting is “Based on signal”. The distance between the planes depends on the value of the process signal. The “Custom signal range” is the default range displayed in the contour plot.

In the context menu of the time axis, there is an option “Auto scale” to scale the time axis so that all planes of that moment are shown.

