FAZT I4 Management API Reference Guide

Author:
Faz Technology

Contents

1 Preface

1.1 Document History

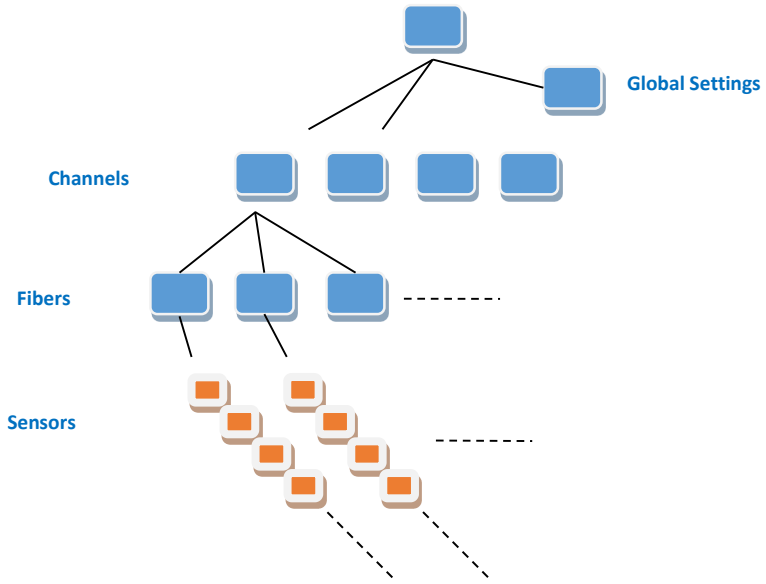
Issue	Date	Authors	Comment
1.0	26 Feb 2016	Faz Technology	Initial release
1.1	31 Mar 2016	Faz Technology	Updated after review.
1.2	12 July 2016	Faz Technology	Updated for Interrogator 1.0.A.8 release.
1.3	18 Aug 2016	Faz Technology	Updated for Interrogator 1.0.A.8 final release

2 Introduction

This document outlines the interface used to manage an Ix interrogator. The API allows the reading and updating of:

- Interrogator settings
- Channel settings
- Fiber settings
- Sensor settings

The settings are organised into a tree-like structure with a channel containing one or more fibers and a fiber containing one or more sensors.



A REST-like interface is used to change the settings in this tree.

3 REST Basics

Only a few basic operations are needed to remotely manage any piece of equipment. For example, take a sensor connected to an Interrogator. Once the sensor has been physically installed, the Interrogator needs to be told about the sensor. Some basic operations are needed to do this e.g.

- add/create a sensor
- fetch the current sensor setting
- update the sensor settings
- delete the sensor

In short, CREATE, READ, UPDATE and DELETE operations. The same operations are potentially needed to remotely manage any other part of the Interrogator.

The REST approach is to re-use the extremely common and easy-to-use HTTP protocol. The HTTP protocol uses REQUESTS, and RESPONSES to those requests. Typical HTTP requests that one might see on the Web are:

	HTTP Request			HTTP Response	
	Method	Path	Body	Header	Body
Send data	POST	/form/survey.js	--- data ----	OK, …	(optional data)
Get data	GET	/images/latest.jpg		OK, …	--- latest.jpg ---
Send data	PUT	/form/survey.js	--- data ----	OK, …	(optional data)
Delete data	DELETE	/docs/sheet.zip		OK, …	

REST uses HTTP to manage remote configuration. For example, to manage one specific control valve (valve 5) in a remote system, the following REST commands might be used:

	HTTP Request			HTTP Response	
REST operation	Method	Path	Body	Header	Body
CREATE	POST	/Valves/5	{ valve settings }	OK, …	{ valve settings }
READ	GET	/Valves/5		OK, …	{ valve settings }
UPDATE	PUT	/Valves/5	{ valve settings (updated) }	OK, …	{ valve settings }
DELETE	DELETE	/Valves/5		OK, …	

So REST uses the HTTP method to indicate what kind of operation to perform. It uses the path to indicate what resource to update e.g. channel 0. And the body is used to carry new settings or retrieve current settings.

REST typically works on a READ-MODIFY-WRITE basis so to fetch and then update the settings for a particular control valve, REST might construct the following HTTP GET and PUT requests …

```
GET     /Valves/5
```

which might return …

```
{ "id": 5, "name": "old_name" }
```

And the following PUT request updates Valve 5 with a new name …

```
PUT     /Valves/5      { "id": 5, "name": "new_name" }
```

Notice here that the HTTP body is in JSON format. JSON format is used in the REST interface to a Faz Ix interrogator.

Also, the REST interface to a Faz Ix Interrogator does NOT support CREATE and DELETE operations.

There are numerous tools for interacting and learning about a REST service e.g. 'Postman' for Chrome, 'RESTClient' for Firefox and 'curl' command line tool for linux.

4 REST usage for Ix interrogator

4.1 REST Operations

The REST API for the Faz Ix interrogator allows a remote client to fetch and update settings for:

- general interrogator operation
- specific channel operation
- specific fiber operation
- specific sensor operation

The REST calls needed for these operations are listed below. Note that the Ix interrogator is case insensitive to the resource path so the path /api/v1/settings is treated the same as /Api/v1/SEttings. The response format sent back for these operations is dealt with later.

Purpose	Method	Path
Read global interrogator settings	GET	/api/v1/Settings
Update global interrogator settings	PUT	/api/v1/Settings
Read settings for all channels	GET	/api/v1/Channels
Read settings for channel 0	GET	/api/v1/Channels/0
Update settings for channel 0	PUT	/api/v1/Channels/0
Read settings for all fibers on channel 1	GET	/api/v1/Channels/1/Fibers
Read settings for fiber 2 on channel 1	GET	/api/v1/Channels/1/Fibers/2
Update settings for fiber 2 on channel 1	PUT	/api/v1/Channels/1/Fibers/2
Read settings for all sensors on channel 1, fiber 2	GET	/api/v1/Channels/1/Fibers/2/Sensors
Read settings for sensor 0 on channel 1, fiber 2	GET	/api/v1/Channels/1/Fibers/2/Sensors/0
Update settings for sensor 0 on channel 1, fiber 2	PUT	/api/v1/Channels/1/Fibers/2/Sensors/0

When updating a resource in the Ix, the HTTP body must contain the new settings for the resource (in json format). There are some rules governing these new resource settings.

1. The resource id in the URI path must agree with the id in the resource settings (json). The resource id is 0 in the following example.

```
PUT /api/v1/Channels/0      {"channelId": 0, "name": "Channel 1",
                             "spectralRate": 4, "spectral": false }
```

2. All read-write resource settings must be supplied during an update, even if only a subset of those settings are being updated. A setting with nested information is the exception to this, mentioned in the next point.

   This also requires that float values, retrieved from the interrogator and subsequently returned in a READ-MODIFY-WRITE sequence, must maintain their original precision. The precision to maintain is 17 significant decimal digits (double precision).

3. Nested updates are not supported.
   In the Ix interrogator, the settings are organised in a tree-like structure. Updating this nested structure in one operation is not supported. This is the reason why the "fibers" setting in channel must be omitted when the channel itself is being updated.

```
PUT /api/v1/Channels/0      {"channelId": 0, "name": "Channel 1",
                             "spectralRate": 4, "spectral": false
                             "fibers": [
```

```
                                    { "fiberId": 0, "name": "xxx", …
                                      "sensors": [ … ] } ]
                                }
```

4. Read-only settings can be omitted during an update operation. If they are included, they are ignored.

```
PUT /api/v1/Settings  {"interrogatorName": "Engine Room", "polarization": "off",
                       "spectrumStart": 1528.5, "spectrumEnd": 1568.0,
                       "triggerSource": "master", …
                      }
```

5. Extra/Unknown settings are ignored. The operation will succeed provided all known parameters are valid.

```
PUT /api/v1/Channels/0      {"channelId": 0, "name": "Channel 1",
                             "spectralRate": 4, "spectral": false,
                             "unknown": value}
```

4.2 REST responses

As with requests, REST responses are HTTP responses. A response contains a Header and an optional Body, separated by a blank line. A REST response from the Ix interrogator indicating a successful operation will have a HTTP status code of 200 and will contain the resource settings in json format. Both READ and UPDATES return the full resource settings in the response. If the request is for a resource with nested resources, then the response will contain all the settings for all nested resources. The following is a basic example of a response to a READ or UPDATE on a channel with no nested fibers.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{ "channelId": 0, "name": "Channel 1",
  "spectralRate": 4, "spectral": false, "fibers": [] }
```

4.3 REST error handling

If a REST request can't be processed, the response sent back will indicate an error. HTTP result/errors codes are used by REST to indicate success/failure. Different HTTP status codes are sent for different error scenarios. The response may optionally include a HTTP response body, in json

format, containing an application error code and an error description. The intention is that this code/description pair should help to clearly identify the error scenario.

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/json
Content-Length: xx

{
        "error":
        {
                "code" : 100,
                "message" : "Item xxx does not exist on Settings."
        }
}
```

The HTTP codes for different scenarios are outlined below:

Event	HTTP Code	Description
Successful Request	200	Standard response for successful requests. Used on GET, PUT, DELETE
Successful Creation	201	Standard response for successful resource creation. Used on POST
URI Syntax Error	400	Error response when request URI contains a syntax error e.g. 'api/Channels/3///'
URI Resource not found	404	Error response for resources that don't exist e.g. 'api/Channels/fff' or 'api/Channels/99' or GET '…/Sensors/0' if sensor 0 doesn't exist
Method not allowed for URI resource	405	Error response when request method is not allowed for resource e.g. DELETE 'api/v1/Mode'
Payload syntax error	400	Error response when HTTP body content is not properly formatted e.g. bad json format – {[..}
Payload semantic error	422	Error response if the HTTP body is syntactically correct but has an error. There are multiple scenarios that can cause a semantic error e.g. - resource id in body conflicts with resource id in URI - mandatory parameter missing - value out of range - nested resources included in a POST/PUT request slave trigger mode or the laser is off
Internal Server error	500	Error response for unforeseen error situations
Method Unknown Method Not Supported	501	Error response when request method not supported e.g. OPTIONS, HEAD

The following web site provides an extended list of HTTP codes for errors and other HTTP responses:
http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

4.4 Concurrent access

The API supports multiple readers and a single writer. More than one client writing values may result in changes to different values being undone. This is because writes typically involve reading a set of values, modifying some and writing back the modified set. If reads and writes from two clients are interleaved, the earlier client write may be overwritten even if the later client does not modify the same values.

5 Detailed API

5.1 Interrogator settings

5.1.1 List of settings

The interrogator will have configuration settings that are required for proper operation. Some settings are for information only (Read-Only) and some can be modified (Read-Write).

Note that two of the Interrogator settings below will disallow any updates to the Interrogator until they are reset:

- laserOn set to "false"
- triggerSource set to "slave"

Setting	Type	Range	Default	Units	Access	Description
interrogatorName	String	-	I4	-	RW	User defined name for Ix interrogator
polarization	String	“on” / “off”	“off”	-	RW	Turn polarization on or off
laserOn	Boolean	true/false	true	-	RO	Indicates if laser button on front panel is on
triggerSource	String	“master” “slave”	“master”	-	RW	Determines if sampling is triggered internally or externally
timeServer	Object	-	-	-	RW	Allows synchronisation of the interrogators time to an NTP server.
type	String	“self” “ntp”	“self”	-	RW	Synchronisation method. “self” indicates no synchronisation.
value	String	Empty / IP addr.	-	-	RW	IP address of the NTP server if type is “ntp”, otherwise ignored.
wavelengthSamplingRate	Double	>0	1000.0	Hz	RO	Ix sampling rate
wavelengthDownSamplingRate	Unsigned	-	1	-	RW	Specify n where every nth sample returned
wavelengthFilterType	String	“none” “butterworth”	“none”	-	RW	Filter to apply to sensor data
wavelengthCutOffPoint	Double	1 up to half samplingRate	480.0	Hz	RW	Cut-off frequency for all sensors

Setting	Type	Range	Default	Units	Access	Description
spectrumStart	Double	-	Ix specific	nm	RO	Start point of spectrum
spectrumEnd	Double	-	Ix specific	nm	RO	End point of spectrum
spectralResolution	Unsigned	-	1	pm	RO	Resolution of spectral data
exclusiveSpectralRate	Unsigned	16	16	Hz	RO	Max spectral sampling rate allowed for a channel (only when one channel is outputting spectral data)
sharedSpectralRate	Unsigned	4	4	Hz	RO	Spectral sample rate when more than one channel is outputting spectral data
minDistance	Double	0 - 15000	0	m	RW	Minimum round trip distance to a sensor. This value offsets the distance range (6500m round trip) in which the interrogator is able to detect sensors.
timingMode	String	“standard” “enhanced” “aligned”	“standard”	-	RW	“standard”: Provides basic sensor timestamps from start of sweep. “enhanced”: Enhanced accuracy of sensor timestamps. Wide FBG sensors (>120pm) are not supported. Lowers total number of sensors supported! “aligned”: Sensor measurements are aligned to the start of sweep. Therefore sensor timestamps are zeroed. Wide FBG sensors (>120pm) are not supported. Further lowers total number of sensors supported!

5.1.2 Detailed Example (GET)

Get the configuration settings for the interrogator - /api/{Version}/Settings

Request:

```
GET /api/v1/Settings
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
  "interrogatorName":"I4",
  "polarization":"off",
  "wavelengthDownSamplingRate":1,
  "spectralResolution":1,
  "wavelengthFilterType":"none",
  "wavelengthCutOffPoint":180.0,
  "wavelengthSamplingRate":1000.0,
  "spectrumStart":1528.5,
  "spectrumEnd":1568.0,
  "exclusiveSpectralRate":16,
  "sharedSpectralRate":4,
  "spectralSamplingRate":16.0,
  "laserOn":true,
  "triggerSource":"master",
  "timeServer": {"type":"self", "value":""},
  “minDistance”:0,
  “timeStamping”:”perSweep”,
  “improvedTiming”:false
}
```

5.1.3 Detailed Example (PUT)

Set the configuration settings for the interrogator - /api/{Version}/Settings

Request:

```
PUT /api/v1/Settings
Content-Type: application/json
Content-Length: xx

{
  "interrogatorName":"I4",
  "polarization":"off",
  "wavelengthDownSamplingRate":1,
  "spectralResolution":1,
  "wavelengthFilterType":"none",
  "wavelengthCutOffPoint":480.0,
  "triggerSource":"master",
  "timeServer": {"type":"self","value":""},
  “minDistance”:10000,
  “timeStamping”:”perSensor”,
  “improvedTiming”:true
}
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
    Same as response to GET but with new values set…
```

```
}
```

5.2 Channel settings

5.2.1 List of settings

The settings for a channel are listed below. The channel id appears in the data stream for sensors connected to this channel. A channel contains a nested structure of fibers and sensors and settings for all nested resources are returned in the response for a READ or UPDATE operation. Two types of sensors can be defined in the Ix interrogator – a 'sensor' which produces a single output value or a 'spectral sensor' which produces a snapshot of a subset of the spectrum (for post processing by a client). The determination of what type of sensors can be added under this channel is controlled at channel level by the 'spectral' setting.

Setting	Type	Range	Default	Units	Access	Description
channelId	Unsigned	0 - 3	-	-	RO	Channel Id. Id's correspond to front panel labels – 1 e.g. CH1 has channelId 0
name	String	-	-	-	RW	User defined name for channel
spectral	Boolean	true/false	false	-	RW	Determines whether normal or spectral sensors can be defined under this channel
spectralRate	Unsigned	4, 16	4	Hz	RW	Rate at which spectral sampling occurs for nested spectral sensors
fibers	Array	-	-	-	RO	Nested fibers and sensors

5.2.2 Detailed Example (GET)

A GET operation on /api/{Version}/Channels will return an array of the full nested information for all channels. For this example, the nested information for a single channel is requested:

Request:

```
GET /api/v1/Channels/0
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
  "channelId":0,
  "name":"Channel 1",
  "spectralRate":4,
```

```
  "spectral":false,
  "fibers":[ .. fiber settings are outlined in separate section … ]
}
```

5.2.3 Detailed Example (PUT)

Request:

```
PUT /api/v1/Channels/0
Content-Type: application/json
Content-Length: xx

{
  "channelId":0,
  "name":"Channel 1",
  "spectralRate":4,
  "spectral":false
}
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
    Same as response to GET but with new values set…
}
```

5.3 Fiber settings

5.3.1 List of settings

The settings for a fiber are listed below. The fiber id appears in the data stream for sensors connected to this fiber. A fiber contains a nested structure of sensors and settings for all nested sensors are returned in the response for a READ or UPDATE operation.

Setting	Type	Range	Default	Units	Access	Description
fiberId	Unsigned	0 - 3	-	-	RO	Fiber Id (unique within a channel)
name	String	-	-	-	RW	User defined name for fiber
sensorProcessing	String	peak / trough	peak	-	RW	Determines if the sensors defined under this fiber have a peak or trough response
defaultDistance	Double	0 – 6500	10.0	m	RW	Default round-trip distance for all sensors on fiber
defaultGain	Unsigned	1 - 4	1	-	RW	Default gain for all sensors on fiber
sensors	Array	-	-	-	RO	List of sensors connected to this fiber

5.3.2 Detailed Example (GET)

A GET operation on /api/{Version}/Channels/0/Fibers will return an array of the full nested information for all fibers for channel 0. For this example, the nested information for a single fiber is requested:

Request:

```
GET /api/v1/Channels/0/Fibers/0
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
  "fiberId":0,
  "name":"Fibre X",
  "sensorProcessing":"peak",
  "defaultDistance":10.0,
  "defaultGain":2,
  "sensors": [ .. sensor settings are outlined in separate section …]
}
```

5.3.3 Detailed Example (PUT)

Request:

```
PUT /api/v1/Channels/0/Fibers/0
Content-Type: application/json
Content-Length: xx

{
  "fiberId":0,
  "name":"Fibre X",
  "sensorProcessing":"peak",
  "defaultDistance":10.0,
  "defaultGain":2
}
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
    Same as response to GET but with new values set…
}
```

5.4 Sensor settings

5.4.1 List of settings

The settings for a sensor are listed below. The sensor id appears in the data stream along with the sensor value. The settings below the dotted line in the table below are not relevant for a spectral sensor and should not appear in either a READ or UPDATE operation.

Setting	Type	Range	Default	Units	Access	Description
sensorId	Unsigned	0 - 31	-	-	RO	Sensor Id (unique within a fiber)
name	String	-	-	-	RW	User defined name for sensor
dataType	String	wavelength / spectral	-	-	RO	Type of sensor
start	Double	spectrumStart - spectrumEnd	-	nm	RW	Start of sensor dynamic range
end	Double	spectrumStart - spectrumEnd	-	nm	RW	End of sensor dynamic range
distance	Double	0 - 6500	-	m	RW	Round-trip distance from interrogator to sensor along fiber.
gain	Unsigned	1 – 4	-	-	RW	Gain level applied to sensor response before sampling
thresholdLevel	Double	0.0 – 3.0	-	-	RW	Minimum sensor response amplitude before processing. Used to eliminate spikes.
thresholdWidth	Unsigned	-	-	freq points	RW	Minimum sensor response width before processing. Used to eliminate spikes.
fitPoints	Unsigned	2 - 1024	-	freq points	RW	Number of raw samples to use when fitting a sensor response
fitAcquired	Boolean	true / false	-	-	RO	Indication whether system managed to successfully fit the sensor response.
data_values	Array[Integer]	Array Size == fitPoints	-	-	RO	Array of spectrum samples of sensor response.
fit_values	Array[Double]	Array Size == fitPoints	-	-	RO	Array of fit values to sensor response.

Note: The 'data_values' and 'fit_values' arrays are only returned when a sensor is UPDATED, and a minimum of one sensor setting is modified.

5.4.2 Detailed Example (GET)

A GET operation on /api/{Version}/Channels/0/Fibers/0/Sensors will return an array of the full nested information for all sensors for fiber 0 on channel 0. For this example, the information for a single sensor is requested:

Request:

```
GET /api/v1/Channels/0/Fibers/0/Sensors/0
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xx

{
  "sensorId":0,
  "name":"Sensor 1",
  "start":1544.12,
  "end":1544.9888682745825,
  "distance":10.0,
  "gain":1,
  "thresholdLevel":0.5,
  "thresholdWidth":10,
  "fitPoints":60,
  "dataType":"wavelength",
  "fitAcquired":false,
  "data_values":[],
  "fit_values":[]
}
```

5.4.3 Detailed Example (PUT)

Request:

```
PUT /api/v1/Channels/0/Fibers/0/Sensors/0
Content-Type: application/json
Content-Length: xx

{
  "sensorId":0,
  "name":"Sensor 1",
  "start":1544.12,
  "end":1544.9888682745825,
  "distance":10.0,
  "gain":1,
  "thresholdLevel":0.5,
  "thresholdWidth":10,
  "fitPoints":60,
  "dataType":"wavelength"
}
```

Success Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
Content-Length: xx

{
    Same as response to GET but with new values set.
    AND data_values and fit_values arrays should be populated.
}
```