json\_API

# Inspection JSON interface

Version 1.06

## Basic concepts

### Return code

All commands returns a JSON which at least one attribute named "status". In the case the command succeeds, "status" is set to the string "ok":

{"status" : "ok"}  
Copy

:label: A successful JSON may contain other attributes.

For deferred jobs, the attribute "status" may hold the state "processing". In this case an additional attribute with the job\_id shall be provided.

{"status" : "processing", "job\_id" : 1234}  
Copy

In the case the command fails, "status" is set to the string "fail" and an additional attribute named "reason" is set to a plain text description in english of the error.  
For simplicity sake, no error codes are given.

{"status" : "fail", "reason" : "not enough memory"}  
Copy

### Job types

The inspection system uses the term "job" to identify an inspection task.

:books: Current job list:

* :one: **"nozzle\_out"** - The nozzle out pattern is captured and processed, returns nozzle out tables.
* :two: **"uniformity"** - The uniformity pattern is captured and processed, returns linearization tables.
* :three: **"uniformity\_check"** - The uniformity pattern is captured and processed, returns measurements for drawing a curve.
* :four: **"camera\_hardware"** - The camera calibration pattern for hardware is captured in several images, one per camera.
* :five: **"camera\_calibrate"\*** - The camera calibration pattern is captured in several images, one per camera. White zone is captured and stored in white reference. Unless protected, stitching is computed.

### Page types

A job may contain a number of pages. Each page may be of a different type. If same page type is scanned twice in the same job, second instance overwrites first.  
This works in this way to allow to repeat wrong scans.

* **"single"** - The one and only page type for this category
* **"C"** - Cyan channel
* **"M"** - Magenta channel
* **"Y"** - Yellow channel
* **"K"** - Black channel
* **"O"** - Orange channel
* **"V"** - Violet channel
* **"W"** - reserved for white
* **"cam1"** - Camera #1
* **"cam2"** - Camera #2
* **"cam3"** - Camera #3
* **"cam4"** - Camera #4
* **"cam5"** - Camera #5

### Products

A product is what the inspection system returns after processing a job. It is labeled with the combination of a page type and a URL for resource location.

Type of data will be returned as MIME type when fetching the resource pointed by the URL. The product comes as part of other JSON structures.

Example 1:

{ "cam1" : "/images/img123.png" }   
Copy

Example 2:

{ "M" : "/products/prod456.json" }  
Copy

## High-Level commands

### Get system status

Returns a system-wide status of all inspection system

{ "command": "system\_status"}  
Copy

Typical return includes many fields:

{   
"status" : "ok",   
"version" : "1.0.0.0",  
"about" : "whatever to show in UI, for example EFI inspection",   
"cameras" : [   
 {   
 "position" : 1,   
 "serial\_number" : "xxx",   
 "model\_name" : "genie",   
 "vendor" : "teledyne",   
 "status" : "ok"   
 },  
 {   
 "position" : 2,  
 "serial\_number" : "yyy",  
 "model\_name" : "genie",  
 "vendor" : "teledyne", "status" : "ok"   
 },  
 etc...  
],  
"light" : {   
 status : "fail",   
 "reason" : "overheat on module #3"   
},  
"current\_job": {  
 "data": { "single": "/products/prod32.json" },  
 "finished": "2024-05-07 17:28:01",  
 "job\_id": 3,  
 "job\_type": "nozzle\_out",  
 "started": "2024-05-07 12:54:17",  
 "status": "ok"  
 }  
 }  
Copy

### Start job:

Sets a job of the given type as the current one.

{ "command" : "start\_job", "job\_type" : "uniformity"}   
Copy

Normally, an accepted job returns a job\_id and a deferred status

{ "status" : "processing", "job\_id" : 1234}  
Copy

### Job Status:

Returns the status of a running job, as well as all associated products in the case the job is completed

{ "command": "job\_status", "job\_id" : 1234 }   
Copy

If the job is still ongoing, the answer reflects such state. Please note job\_id is echoed as an ack.

{ "status" : "processing", "job\_id" : 1234 }  
Copy

It is possible to omit job\_id. In this case the status of the active job and its ID are returned.

If the job is complete, the status JSON may contain a reference to the obtained products. In this case a single table. All products can be retrieved a by issuing a GET on the http(s) server on the given URL.  
:label: see Product definition

{ "status" : "ok", "job\_id" : 1234, "data" : { "table" : "/products/product4567.txt" } }   
Copy

In this case multiple tables

{ "status" : "ok", "job\_id" : 1234, "data" : [ "C" : "/products/product4568.txt", "M" : "/products/product4568.txt" ] }   
Copy

Products can be images too:

{ "status" : "ok", "job\_id" : 1234, "data" : { "single" : "/images/img1234.png" } }  
Copy

In the case the job failed, status returns the usual error report

{ "status" : "failed", "reason" : "too much skew"}   
Copy

## Software workflow

This workflow how an external program can drive the inspection system.

Step 1: Send a "start\_job" JSON to put the inspection system in on-guard state.

**Request**:

{ "command" : "start\_job", "job\_type" : "uniformity"}   
Copy

Step 2: Inspection shall create a job and return its job-id

**Response**

{  
 "job\_id": 56,   
 "job\_type": "uniformity",  
 "started": "2024-06-11 09:40:18",  
 "status": "processing"  
}  
Copy

Step 3: Poll inspection by sending "job\_status" commands until the status reaches the "ok" state.

**Request**:

{ "command" : "job\_status", "job\_id" : 56 }  
Copy

Step 4: When status reaches "ok", use GET to download any products and forward them to Fiery RIP.

**Response**

{  
 "data": {  
 "M": "/products/prod1.json",  
 "single": "/products/prod6.jpg"   
 },  
 "finished": "2024-06-11 09:41:26",  
 "job\_id": 56,  
 "job\_type": "uniformity",  
 "started": "2024-06-11 09:40:18",  
 "status": "ok"  
}  
Copy

In this case, product is the uniformity table for channel magenta as a JSON.  
There is an extra product labeled "single" which is a debug image intended for web.  
Press software should ignore any products other that what is documented for Fiery communication.

### Error handling

On any error, the job\_status command returns "failed" as state and an attribute "reason" is returning with a string describing the error.  
Sending a job when any other is still active cancels the previous job. There is no timeout so a job may rest forever waiting completion.

### Clean queue of jobs:

Erases all pending jobs. Intended to provide a hard-reset with cleanup. **Use with care!**

{ "command" : "clean\_jobs"}   
Copy

## Lights

Light is fully controlled by the framework and external apps should not change its behavior. Some command are provided, however, for debug reasons.

### Turn light ON/OFF

Override the light mechanism to turn it on or to force it off no matter its elapsed time. Useful for camera calibration and factory related operation.

{ "command" : "turn\_light", "data" : "on" }   
{ "command" : "turn\_light", "data" : "off" }   
Copy

Please note turning light ON for extended time lapses can be harmful for light modules.

### Turn light tower :

Override the light mechanism to turn it on or to force it off no matter its state. Useful for debug.

{ "command" : "turn\_light\_tower", "light\_num" : 1, "data" : "on" }   
Copy

## Settings

### Get all configuration:

Use this command to retrieve info on all settings. Settings may hold following types:

* boolean: **"boolean**
* integer: **"integer"**
* real number: **"real"**
* string: **"string"**
* list of bools: **"boolean\_list"**
* list of integers: **"int\_list"**
* list of real numbers: **"real\_list"**
* list of strings: **"string\_list"**

Each setting belongs to a group indentified by an integer. The command can optionally filter settings that belongs to a given group.

Example 1:

{ "command" : "get\_all\_config"}  
Copy

Example 2:

{ "command" : "get\_all\_config", "group" : 1 }   
Copy

The configuration is returned using following syntax (no group filtering is done in this sample).

{  
 "data": [  
 {  
 "data": "c:\\ProgramData\\EFI\\inspection\\database.db",  
 "group": 0,  
 "help": "Full path of database file in the server",  
 "name": "database",  
 "prompt": "Database path",  
 "type": "string"  
 },  
 {  
 "data": 90,  
 "group": 0,  
 "help": "Intensity of the LED light in percent",  
 "name": "light\_intensity",  
 "prompt": "Light intensity",  
 "type": "integer"  
 },  
 {   
 "group": 0,  
 "help": "How many seconds the light remains before getting dimmed",  
 "name": "light\_persistence\_seg",  
 "prompt": "Light persistence (secs)",  
 "type": "integer\_list"  
 "data": [  
 500,  
 550,  
 500,  
 560,  
 600  
 ]  
 }  
 ],  
 "status": "ok"  
}  
Copy

### Get setting:

Use this command to retrieve a given setting identified by name

{ "command":"get\_config", "name" : "stitching\_offset" }   
Copy

### Set setting:

Use this command to set a specific setting identified by name. Type must match.

{ "command":"set\_config", "name" : "stitching\_offset", "data" : 23456.7 }   
Copy

### Black capture

This command measures the black level of each pixel for FFC.

{ "command" : "capture\_cameras\_black", "covered\_cameras" : [1, 2, 3] }   
Copy

Data holds a list on which cameras to capture. If missing, the setting "covered\_cameras" is used.

### White capture

This command measures the white level of each pixel for FFC. A white paper must be placed below camera.  
Always uses all cameras

{ "command" : "capture\_cameras\_white" }   
Copy

### Stitching

#### **"compute\_stitching"**

After a stitching job has been processed, the job products are an image per camera plus the stitched image.  
Camera images comes tagged with page types "cam1" ... "cam5", stitched image comes tagged as "single"

It is possible to change the position of stitching points and recomputing the stitched image. This does not require to send another job but it is neccesary the last job used were a **"camera\_calibrate"**.

{  
 "command" : "compute\_stitching",   
 "cam1" : [[1,2], [3,4], [5,6], [7,8] ...],   
 "cam2" : [[1,2], [3,4], [5,6], [7,8] ...],   
 "cam3" : [[1,2], [3,4], [5,6], [7,8] ...],   
 "cam4" : [[1,2], [3,4], [5,6], [7,8] ...],   
 "cam5" : [[1,2], [3,4], [5,6], [7,8] ...]   
}  
Copy

Numbers are (x, y) of 4 stitching points, per camera.

When successful, it returns the status of the **"camera\_calibrate"** job with all attached products.

#### **"get\_stitching\_points"**

Returns the current stitching points as stored in the configuration.  
Format is same as compute\_stitching. "data" is optional

**Request**:

{ "command" : "get\_stitching\_points", "data" : "cam2" }   
Copy

**Return**:

{   
 "status" : "ok",  
 "disabled": [false, false, false, false, false],  
 "data": [  
 "cam1" : [[1,2], [3,4], [5,6], [7,8] ],   
 "cam2" : [[1,2], [3,4], [5,6], [7,8] ],   
 "cam3" : [[1,2], [3,4], [5,6], [7,8] ],   
 "cam4" : [[1,2], [3,4], [5,6], [7,8] ],   
 "cam5" : [[1,2], [3,4], [5,6], [7,8] ]  
 ]  
}  
Copy

**Example 1** (request):

{ "command" : "get\_stitching\_points", "data" : "cam2" }   
Copy

**Example 1** (response):

{   
 "status" : "ok",  
 "disabled": [false,false,false,false,false],  
 "data": [  
 "cam2" : [ [1,2], [3,4], [5,6], [7,8] ]  
 ]  
}  
Copy

**Example 2** (request)

{ "command" : "get\_stitching\_points", "data" : "none" }  
Copy

**Example 2** (response)

{  
 "data": null,  
 "disabled": [false,false,false,false,false],  
 "status": "ok"  
}  
Copy

#### **"detect*stitching\_points*"**

To force point detection, use this command. No stitching is calculated, only stitching points are returned.  
job\_id is optional and if used, refers to camera calibration job to retrieve camera image. By default is last job.

{  
 "command" : "detect\_stitching\_points",   
 "job\_id" : 14,  
 "data" : "cam2"  
}  
Copy

On camera 0 an additional property **left\_margin** will be returned. On cam 4 (last) **right\_margin** will be returned as well.

#### Shutting down windows on inspection blade

Use following command to shutdown windows (needs permissions)

{"command" : "shutdown"}  
Copy

## Camera calibration workflow (Used by webpages)

### **FCC calibration:**

* Select which cameras are covered, set **"covered\_cameras"** setting or read it into a list.
* Send a **"capture\_cameras\_black"** JSON command.
* Uncover all cameras
* Send a **"capture\_cameras\_white"** JSON command.

### **Calibration**

This step measures white and stitching information for each camera.

* Start a **"camera\_calibrate"** job from press software.
* When completed, you got one image per camera as product
* you can fine tune points by sending **"compute\_stitching"**

### **Save**

* This step forces to recalculate everything when user has modified parameters.

WEBPAGE 3rdParty

<https://xxjapp.github.io/xdialog/>