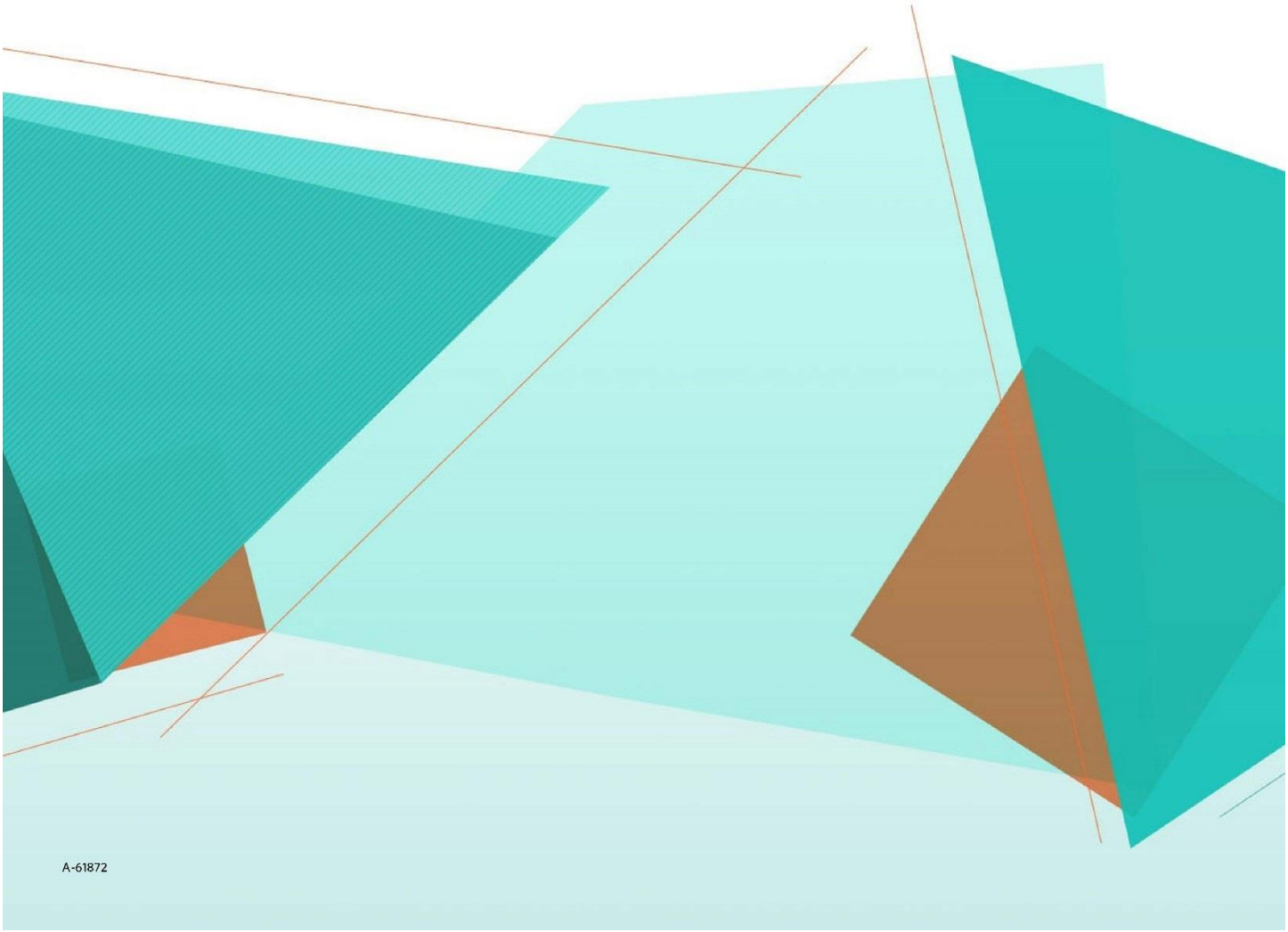
Alaris

Web API Guide for
Kodak Alaris Network Enabled Scanners



Public

Contents

Introduction 3

References 3

HTTP in the APIs 3

Reference Section 5

SCANNER API 5

[method] api/scanner/<function> 5

Scanner Status 6

Scanner Capabilities 7

Scanner Log 14

CONFIG API 15

[method] api/config/<function> 15

Configuration Settings 15

Get Image of scanner 17

Upload SSL/TLS Certificate 17

SESSION API 20

[method] api/session/<function> 20

Session Create 20

Session End 22

Session Status 23

Session Status – Status Only (Added in FW v1.07) 27

Session Status – Configuration Only (Added in FW v1.07) 28

Session Configuration 29

Session Configuration rules 32

Session Recreate and Scan 34

Session Start Scan 34

Session Stop Scan 34

Session Get Metadata 35

Session Get Image 37

Session Delete Image (Added to S2000w as of Firmware 180902) 38

Session Get Meters (Added to S3000 as of Firmware v220203) 39

CORS support 40

To Turn on CORS and CORS Preflight support: 40
CORS Header response 40
CORS Preflight 40
Error Codes – HTTP 41
Error Codes – returned from Status 43
ErrorNum Table 45
Timeouts 47
More Details 48
Scan Example 48
Scan API: URL and JSON Response 49
Programmers Page to play with the protocol 51
Diagnostic / Test page 51
Scan Example, pseudo code: 52
Typical Sequence to use the API's 53
Scanning 53
To scan many pages 54
Security 54
Why use the Scanner API vs the Session API ? 55
How to determine the connection type from the IP 55
Discovery Protocols 56
TXT Record from Scanner for _scanner service: 56
DNS Address 56
Link-Local Address 56
Static IP 56
Diagram Scanning States 57
Normal Scan 57
Second Job Scan in same Session 58
Skip Blank Pages – Back side of first sheet is blank, two sheets 59
Fiddler capture of scan 60
Fiddler's Composer Scratchpad 61

Public

Introduction

This document provides definition of the HTTP v1.1 Application Programming Interfaces for the Kodak Alaris Network Enabled Scanners:

- i11x0WN
- S2000 Series
- S3000 Series

The Introduction contains concepts and terminology that will be used throughout the document. The Scanner API and the Session API control the functionality of scanning and monitoring progress. They are explained in this document. The Configuration API is referenced, but it is not covered in depth.

The APIs use a RESTful protocol for accessing the scanner functionality through HTTP commands. The scanner driver does not need to be installed on your host PC for you to access the API. All web browsers support a language that can easily communicate via a RESTful protocol. Many compliable languages, and interpretive languages also talk RESTful protocols. Without installing any driver on your host PC, for any operating system, this API is accessible.

A STATEFUL protocol is implemented during scanning. A session must be created. During the session, no other application on a different PC can interrupt the current session. They will receive an error when they attempt to create their own session. They can use the Scanner API to read information about the scanner and the current state of the scanner.

Use the http://<ip>/conf/diag.html page to play with the individual commands and to get used to how this works before writing code. Please read up on how Status is required to drive the states of the scanner during scanning.

References

HTTP Protocol	http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
Error Codes	https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP in the APIs

Commands

HTTP is the language of the web. Our APIs define a HTTP language to drive our scanners. Our APIs use the following commands:

- GET – reads information from the scanner, but does not change any information, state, or start any work.
- POST – starts (creates) a process and creates the SessionId. The SessionId is used on every function within the API to assure you are the only one accessing the data.
- PUT – changes (updates) the state of a running process.
- DELETE – removes the current process that was started with POST.

PUT and POST require JSON value pairs to define the command parameters. The parameters can be put either in the HTTP body or in the URL of the HTTP command.

$Variable references below define a string or number variable that is set by the caller.

Public

All data responses are in JSON format.

All commands (“startscan”, “status” etc.) are case **in**sensitive, e.g: “StartScan” = “STARTSCAN”.

Addressing your scanner

Examples in this document show 10.0.0.1 as the IP address, which is the default ad hoc IP address (also referred to as Wireless Directly in other scanner documentation). When using the API, you should use the actual IP address for your scanner (see the Discovery section).

The APIs support both HTTP, port 80 and HTTPS, port 443, to access the scanner web pages and internal web API. Port 80 is the default port for the web server and is used by the scanner for HTTP. Note that this port number will be advertised with the discovery service. Port 443 is used for HTTPS, TLS secure access to the scanner APIs and web site.

Read the IP address and port from the discovery service. Use those to resolve the scanner address.

Public

Reference Section

SCANNER API

The SCANNER API is read only and enables monitoring of the state of the scanner, and various information about the scanner. The following functions are available for the Scanner APIs:

[method] api/scanner/<function>

Method	Function	Description
GET	Status	Retrieve the scanner status; can be performed at any time.
GET	Capabilities	Read capabilities of the API, which can be used to learn the language and syntax of the commands.
GET	Log/{Optional scan log name}	Obtain the scanner logs from the scanner.

Public

Scanner Status

Method: GET Function: **Status**

Verify information about the scanner, and its current state.

[GET] https://10.0.0.1/api/scanner/status or use HTTP

Params in body: none

Extra HTTP Header Values: none

Provides scanner information useful for determining if the scanner is accessible, and what type of scanner this is.

Return Value: ("Variable":Value, Variables are case insensitive)

Returned in JSON format the variables to show the scanner status.

Info:

Vid – Vendor ID
Pid – Product ID
Mfg - Manufacturer
Mdl – Model
ScannerState can be (Idle, In Session, Scanning, Done Scanning, Error) <- See Scanner Capabilities
ErrorNum= 0 <- Scanner Error num. NOT last HTTP error. Is a status of the embedded scanner.
Displayname – intended use as Friendly name for display
HostName – Fixed string, may be different from Displayname, this is the name of the scanner and used in DNS addressing. ie: <HostName>.domain
SerialNum – Fixed string – unique identifier of this scanner
EstimatedMaxNumImages=50 <- currently fixed, may be calculated in the future
OCPUserName="Vinnie" <- Who's currently scanning

Raw HTTP Command:

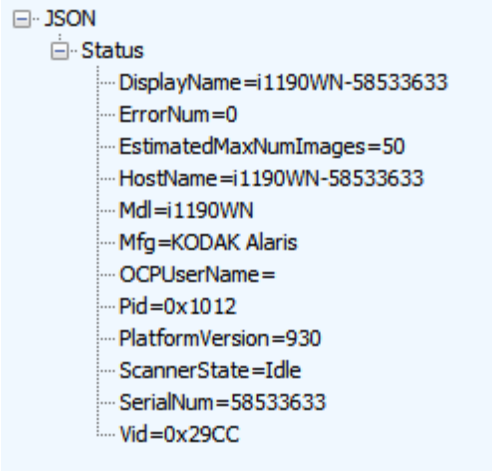
GET https://10.0.0.1/api/scanner/status

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Length: 331
Content-Type: application/json; charset=utf-8
X-Frame-Options: DENY
Date: Thu, 10 May 2018 20:16:14 GMT
Server: lighttpd/1.4.49

{"Status":{
        "Vid":"0x29CC",
        "Pid":"0x101C",
        "Mfg":"KODAK Alaris",
        "Mdl":"S2080w",
        "ScannerState":"Done Scanning",
        "ErrorNum":0,
        "EstimatedMaxNumImages":50,
        "DisplayName":"S2080w-64633610",
        "HostName":"S2080w-64633610",
        "SerialNum":"64633610",
        "PlatformVersion":"180506",
        "OCPUserName":""}}
```

Graphical view of JSON response:



Public

Scanner Capabilities

Read capabilities of the API, which can be used to learn the language and syntax of the commands. It is intended to enable you to learn what has changed from release to release, dependencies and syntax of commands.

Method: GET Function: **Capabilities**

[GET] https://10.0.0.1/api/scanner/capabilities <- V1 capabilities

[GET] https://10.0.0.1/api/scanner/capabilities?V1

[GET] https://10.0.0.1/api/scanner/capabilities?V2

Params in body: none

Extra HTTP Header Values: none

Return Value:

V1 Capabilities:

Reverted in our 190903 release to require specifying V2 to obtain V2 capabilities. The scanner always provides V2 capabilities but will return V1 or V2 information to maintain compatibility with older applications that know only about V1 functionality. This shows a list of commands that could be configured in the PUT of api/session to change the configuration of the scanner.

V2 Capabilities*:*

The V2 capabilities structure reveals more functionality provided within the scanner and a reorganization of capabilities to enable dynamic menu design, once you understand how this work. It should be obvious how this functionality works, but we'll provide an explanation below

A list of JSON data that contain three primary sections:

Configuration is an array of strings defining the key and values that are allowed in the PUT api/session command, to configure a scan job.

Since many primary controls have additional fields that are only valid when a selection is made, we added SubControls. For example, DPI has no subcontrols but if you choose AutomaticAdvanced for ForegroundboldnessAggressiveness, you would show a slider selection with an English label of "Aggressiveness", and a range of -10 to 10.

Note: We only support English in the text for this feature.

MaxDocumentLength has no primary control to configure it, we choose to show "#NoPrimaryControl#". This means if you are dynamically creating UI items, you should skip this control, but look in the SubControls for the corresponding primary control to determine if you should populate a sub control item for each item.

SubControls: this provides a mechanism to enable a look up for every Configuration value to determine if this is another setting for that value. For example: ColorDropOut when set to Predominant as a Subcontrol: SubControls->Predominant->ColorDropOut that defines a command of "ColorDropOutAggressiveness" with a minimum value of -10 and maximum of 10, and a label named "Aggressiveness". Then if you look in the defaults section you can see that the default is 0.

Public

Defaults: this section match's what you will get the first time you create a session and read the api/session/configuration. It provides the list of variables that can be set, and their default values. It does not tell you of the dependency on those variables. But you could send "Defaults" in a "Configuration" in an api/scanner PUT call to force defaults or make single entry modifications to settings in the scanner. All settings can be applied, even those inactive due to, for example, the color mode. So set ColorDropOut, but it won't take effect until the ColorMode is changed to BW. This enables you to read all settings, change one or more, and put them all back to the scanner without error.

Current List:

Current capabilities list as of July 2023:

Scan Feature	Available Values	Scanner Support
DPI	• 100 (S3000 Series Only) • 150 (S3000 Series Only) • 200 • 300 • 600 (S3000 Series Only)	• i11x0WN • S2000 Series • S3000 Series
ScanSide	• Simplex • Duplex	• i11x0WN • S2000 Series • S3000 Series
ColorMode	• Color • Gray • BW • Color_BW (S2000 and S3000 Only)	• i11x0WN • S2000 Series • S3000 Series
SkipBlankPages	• 0 • 1	• i11x0WN • S2000 Series • S3000 Series
SkipBlankPagesContent	• 0 to 100	• S3000 Series
AutoStart	• 0 • 1	• i11x0WN • S2000 Series • S3000 Series
ColorDropout	• None • Red • Green • Blue • Orange • OrangeAndRed • Predominant • Multiple	• i11x0WN • S2000 Series • S3000 Series
ColorDropOutAggressiveness	• -10 to 10	• S2000 Series • S3000 Series
OutputType	• Images • SinglePageColorPdfPlus2Tifs	• S2000 Series • S3000 Series

Public

MaxDocumentLength	• 25 to 400	• S2000 Series • S3000 Series
ColorAutoBrightnessMode	• None • Automatic	• S2000 Series • S3000 Series
ColorBalanceMode	• None • Manual • Automatic • AutomaticAdvanced	• i11x0WN • S2000 Series • S3000 Series
ColorBalanceAggressiveness	• -2 to 2	• S2000 Series • S3000 Series
ColorBalanceRed	• -50 to 50	• S2000 Series • S3000 Series
ColorBalanceGreen	• -50 to 50	• S2000 Series • S3000 Series
ColorBalanceBlue	• -50 to 50	• S2000 Series • S3000 Series
ForegroundBoldnessMode	• None • AutomaticAdvanced • Automatic	• S2000 Series • S3000 Series
ForegroundBoldnessAggressiveness	• -10 to 10	• S2000 Series • S3000 Series
BackgroundSmoothingMode	• None • AutomaticAdvanced • Automatic	• S2000 Series • S3000 Series
BackgroundSmoothingAggressiveness	• -10 to 10	• S2000 Series • S3000 Series
BinarizationMode	• iThresholding	• S2000 Series • S3000 Series
BinarizationContrast	• -50 to 50	• S2000 Series • S3000 Series
MultifeedSensitivity	• None • Low • Medium • High	• S3000 Series
MultifeedResponse	• Stop • StopAndGenerateImages	• S3000 Series
PostScanRotation	• None • Automatic • Automatic90 • Automatic270	• S3000 Series
EdgeFill	• None • Automatic	• S3000 Series

Public

	• AutomaticIncludeTears	
ScanSource	• Automatic (S2000 Series Only) • DocumentFeeder • Flatbed (S2000 Series Only) • Elevator100 (S3000 Series Only) • Elevator300 (S3000 Series Only) • Automatic100 (S3000 Series Only) • Automatic300 (S3000 Series Only)	• S2000 Series • S3000 Series
ImageBorder	• None • Add • Remove	• S3000 Series
TransportHandling	• Normal • ImprovedStacking • BestStacking • Special	• S3000 Series
JpegQuality	• Draft • Good • Better • Best • Superior	• S3000 Series
DocumentFeederTimeout	• 0 to 120	• S3000 Series
DocumentFeederTimeoutResponse	• Stop • Pause • PauseAndWait	• S3000 Series
HoleFill	• 0 • 1	• S3000 Series

Implementors Tips:

The response to this command should be analyzed by developers to determine what changes have been made to the API.

Note – we now do validation checking on all numeric fields. If you do “AutoStart”:”0” – previously we ignored the “0” and this worked, but now we will return a 406 to inform you that the format is incorrect, with additional HTML in the body and in the X-Status header to tell you exactly what is wrong. If you do, for example: “Duplex”: Single, which is missing the quotes, you’ll get a 405 informing you the character offset within the JSON data that is not proper JSON.

The Capabilities command is not needed within your application but is very important to a developer to learn what has been added, and details on how to learn the API. You should do a quick GET on <ip>/api/scanner/capabilities to see what is new in the scanner, after updating firmware. More features may have been added, which will become evident from calling Capabilities. Also, Capabilities will enable you to know exactly the parameters, range, defaults supported for each scan setting. The way this is structured is that primary commands are a list of controls with an array of choices. Each choice may have a linked “SubControl” that compliments the primary command. The SubControl is one or more

Public

keys where the values are integer. Other than the specific commands and subcommands what you cannot tell is the dependency on the color mode. The rules for those dependencies must be found in this document. For example: Scanning B&W TIFF, and Color PDF (Dual Stream), requires OutputType to be set to Color_BW. The http://<ip>/conf/diag.html page and underlying JavaScript may help, but also look at Session Configuration rules.

Our goal is to not remove any features in the existing API, so the code that you develop for the very first version of the firmware, will work in subsequent firmware releases. For example, AutoStart was not provided in the first release of the Web API. It was added and can be configured via the PUT configuration command, but if not included will default to the original setting of 0 or off, which is the default setting after the addition. The same holds true for all of the new features.

Capabilities documents what you can submit as a job configuration via the PUT api/session/configuration command. To confirm a feature is available within the scanner, you could implement within your code to validate it via this function, and flag if it is not listed. Some scanners may support certain features depending on the firmware version. Tracking the firmware version via api/config/settings.

Raw HTTP Command:

GET https://10.0.0.1/api/scanner/capabilities or ?V1

Raw HTTP Response to V1:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, X-Requested-With, Origin, Content-Length, SessionId
Content-Length: 383
Content-Type: application/json; charset=utf-8
X-Frame-Options: DENY
Date: Fri, 20 Sep 2019 16:30:46 GMT
Server: lighttpd/1.4.49
{"Configuration":{
  "DPI":[200,300],
  "ScanSide":["Simplex","Duplex"],
  "SkipBlankPages":[0,1],
  "ColorMode":["Color","Gray","BW"],
  "State":["Idle","In Session","Scanning","Done Scanning","Error"],
  "AutoStart":[0,1],
  "ColorDropOut":["None","Red","Green","Blue","Orange","OrangeAndRed","Predominant","Multiple"],
  "OutputType":["Images"]
},
"Version":1
}
```

Raw HTTP Command:

GET https://10.0.0.1/api/scanner/capabilities?V2

Raw HTTP Response V2:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, X-Requested-With, Origin, Content-Length, SessionId
Content-Length: 2844
Content-Type: application/json; charset=utf-8
X-Frame-Options: DENY
Date: Fri, 20 Sep 2019 16:33:34 GMT
Server: lighttpd/1.4.49


{
```


Public

```
  "Configuration": {
    "DPI": [100,150,200,300,600],
    "ScanSide": ["Simplex","Duplex"],
    "SkipBlankPages": [0,1],
    "SkipBlankPagesContent": ["#NoPrimaryControl#"],
    "ColorMode": ["Color","Gray","BW","Color_BW"],
    "State": ["Idle","In Session","Scanning","Done Scanning","Error"],
    "AutoStart": [0,1],
    "ColorDropOut": ["None","Red","Green","Blue","Orange","OrangeAndRed","Predominant","Multiple"],
    "OutputType": ["Images","SinglePageColorPDFPlus2TIFs"],
    "ColorAutoBrightnessMode": ["None","Automatic"],
    "ColorBalanceMode": ["None","Manual","Automatic","AutomaticAdvanced"],
    "ForegroundBoldnessMode": ["None","AutomaticAdvanced","Automatic"],
    "BackgroundSmoothingMode": ["None","AutomaticAdvanced","Automatic"],
    "BinarizationMode": ["iThresholding"],
    "MaxDocumentLength": ["#NoPrimaryControl#"],
    "MultifeedSensitivity": ["None","Low","Medium","High"],
    "MultifeedResponse": ["Stop","StopAndGenerateImage"],
    "PostScanRotation": ["None","Automatic","Automatic90","Automatic270"],
    "EdgeFill": ["None","Automatic","AutomaticIncludeTears"],
    "ScanSource": ["DocumentFeeder","Elevator100","Elevator300","Automatic100","Automatic300"],
    "ImageBorder": ["None","Add","Remove"],
    "TransportHandling": ["Normal","ImprovedStacking","BestStacking","Special"],
    "JpegQuality": ["Draft","Good","Better","Best","Superior"],
    "DocumentFeederTimeout": ["#NoPrimaryControl#"],
    "DocumentFeederTimeoutResponse": ["Stop","Pause","PauseAndWait"],
    "HoleFill": [0,1]
  },
  "SubControls": {
    "Manual": {
      "ColorBalanceMode": {
        "ColorBalanceRed": {"Min": -50,"Max": 50,"Label": "Red"},
        "ColorBalanceGreen": {"Min": -50,"Max": 50,"Label": "Green"},
        "ColorBalanceBlue": {"Min": -50,"Max": 50,"Label": "Blue"}
      }
    },
    "AutomaticAdvanced": {
      "ForegroundBoldnessMode": {"ForegroundBoldnessAggressiveness": {"Min": -10,"Max": 10,"Label": "Aggressiveness"}},
      "BackgroundSmoothingMode": {"BackgroundSmoothingAggressiveness": {"Min": -10,"Max": 10,"Label": "Aggressiveness"}},
      "ColorBalanceMode": {"ColorBalanceAggressiveness": {"Min": -2,"Max": 2,"Label": "Aggressiveness"}}
    },
    "Predominant": {
      "ColorDropOut": {"ColorDropOutAggressiveness": {"Min": -10,"Max": 10,"Label": "Aggressiveness"}}
    },
    "Multiple": {
      "ColorDropOut": {"ColorDropOutAggressiveness": {"Min": -10,"Max": 10,"Label": "Aggressiveness"}}
    },
    "iThresholding": {
      "BinarizationMode": {"BinarizationContrast": {"Min": -50,"Max": 50,"Label": "Contrast"}}
    },
    "#NoPrimaryControl#": {
      "MaxDocumentLength": {"MaxDocumentLength": {"Min": 25,"Max": 400,"Label": "Maximum Document Length (0.1 inches)"}},
      "SkipBlankPagesContent": {"SkipBlankPagesContent": {"Min": 0,"Max": 100,"Label": "Maximum Page Content Required to Skip"}},
      "DocumentFeederTimeout": {"DocumentFeederTimeout": {"Min": 0,"Max": 120,"Label": "The Amount of Time to Run the Feeder Before Performing the
Timeout Response"}}
    }
  },
  "Defaults": {
    "DPI": 200,
```


Public

```
    "ScanSide": "Duplex",
    "ColorMode": "Color",
    "SkipBlankPages": 0,
    "SkipBlankPagesContent": 5,
    "AutoStart": 0,
    "ColorDropOut": "None",
    "ColorDropOutAggressiveness": 0,
    "OutputType": "Images",
    "ColorAutoBrightnessMode": "Automatic",
    "ColorBalanceMode": "Automatic",
    "ColorBalanceAggressiveness": 0,
    "ColorBalanceRed": 0,
    "ColorBalanceGreen": 0,
    "ColorBalanceBlue": 0,
    "ForegroundBoldnessMode": "None",
    "ForegroundBoldnessAggressiveness": 0,
    "BackgroundSmoothingMode": "None",
    "BackgroundSmoothingAggressiveness": 0,
    "BinarizationMode": "iThresholding",
    "BinarizationContrast": 0,
    "MaxDocumentLength": 140,
    "MultifeedSensitivity": "None",
    "MultifeedResponse": "Stop",
    "PostScanRotation": "Automatic",
    "EdgeFill": "Automatic",
    "ScanSource": "Elevator300",
    "ImageBorder": "None",
    "TransportHandling": "Normal",
    "JpegQuality": "Good",
    "DocumentFeederTimeout": 2,
    "DocumentFeederTimeoutResponse": "Stop",
    "HoleFill": 0
  },
  "Version": 2
}
```


Public

Scanner Log

Method: GET Function: **Log**

[GET] https://10.0.0.1/api/scanner/log<? logfilename.ext>

<optional parameter> - if missing the logfilename will be "`S2000 Series Scanner.log`". The text file downloaded will be stored in the default download folder from a browser. Logs won't be overwritten. New logs will append (1), or (2) …

Params in body: none

Extra HTTP Header Values: none

Return Value:

Text document.

Raw HTTP Command:

https://10.234.10.175/api/scanner/log

https://10.234.10.175/api/scanner/log?LogFile.txt

Raw HTTP Response:

```
HTTP/1.1 200 OK
X-Frame-Options: DENY
content-disposition: attachment; filename=S2000 Series Scanner.log
Content-Length: 96057
Date: Wed, 17 Feb 2016 08:13:14 GMT
Server: lighttpd/1.4.35

<text information contained in various system logs of our product>
```


Public

CONFIG API

This is a small sampling of the configuration API which is available to fully configure the connectivity of the scanner.

[method] api/config/<function>

Method	Function	Description
GET	Settings	Returns a block of configuration settings and current network settings. S2000w has values from scanner/status here to reduce the number of queries.
GET	Image?Scanner	Returns a PNG image of the scanner you are attached to. This image will have transparent background, so can be scaled to fit on any color background you have on your web page.
POST	Certificate?SERVER_CERT	Uploads an SSL/TLS certificate to the scanner for secure HTTPS communication.

Configuration Settings

Method: GET Function: **Settings**

[GET] https://10.0.0.1/api/config/settings

Params in body: none

Extra HTTP Header Values: none <- since there is no SessionId, this can be issued by anyone at any time.

Return Value:

A JSON response that contains all of the configuration settings and active settings (IP address…).

Implementors Tips:

Additional values may be added to this list. All errors will follow the normal HTTP error responses (200=success). An additional Status field is put in the header response to provide additional information for any errors that result in a duplicate number. All errors will be documented.

SystemStatus.PlaformVersion will be important. "181005" is the first release with this enhanced version of the Capabilities command. The version is formatted YYMMRR where RR is an increasing value and captures multiple releases within the same month. It is not the day.

Note that in most cases you cannot tell whether you are connected via the network to a device that is using either WiFi or Ethernet. But via these settings you can determine how this scanner is configured.

WIFI_MODE – supports values of "STA", "OFF" or "AP"

STA – stands for Station, otherwise known as Infrastructure, which we refer to as wireless network

OFF – WiFi is turned off

AP – WiFi is in ad-hoc or standalone network mode which we call Wireless Directly

ETH0_IP_MODE – supports values of "DHCP", "STATIC" or "OFF"

DHCP is a normal Ethernet cable that obtains it's IP from a server.

STATIC is a manually configured IP address.

OFF means Ethernet is not configured.

ETH0_CARRIER_DETECTED – even though Ethernet may be detected "1", it may not be configured.

You can even detect if USB is connected: USB_HOST_CONNECTED.

Public

You could choose to reconfigure a scanner on the fly if desired, by detecting that you are in a WIFI Mode and reconfigure it for Ethernet if an Ethernet cable is detected.

Raw HTTP Command:

https://10.0.0.1/api/config/settings <- where 10.0.0.1 is the IP address of the scanner

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Length: 1331
Content-Type: application/json; charset=utf-8
X-Frame-Options: DENY
Date: Thu, 10 May 2018 20:30:06 GMT
Server: lighttpd/1.4.49

{"SystemStatus":
{
	"USB_HOST_CONNECTED":"0",
	"ETH0_CARRIER_DETECTED":"1",
	"ETH0_IP_ADDRESS":"10.234.10.17",
	"WIFI_IP_ADDRESS":"10.0.0.1",
	"ETH0_NETMASK":"255.255.255.0",
	"WIFI_NETMASK":"255.255.255.0",
	"ETH0_MAC_ADDRESS":"c8:fd:19:2b:10:fc",
	"WIFI_MAC_ADDRESS":"3c:a0:67:44:af:8e",
	"WIFI_SIGNAL_STRENGTH":"4",
	"WIFI_LINK_QUALITY":"4",
	"WIFI_BITRATE":"72.2",
	"WIFI_ESSID":"S2080w%2D64633610",
	"LOW_POWER_MODE":"0",
	"FLATBED_CONNECTED":"0",
	"SW_UPDATE_STATUS":"",
	"NUM_PROCESSOR_CORES":"2",
	"NUM_DSP_CORES":"2",
	"CPU_TEMPERATURE":"60200",
	"GPU_TEMPERATURE":"59800",
	"CORE_TEMPERATURE":"58500",
	"DSP_TEMPERATURE":"59800",
	"IVA_TEMPERATURE":"59800",
	"WIFI_MODE":"AP",
	"ETH0_IP_MODE":"DHCP",
	"WIFI_STA_IP_MODE":"DHCP",
	"WIFI_AP_IP_ADDRESS":"10.0.0.1",
	"WIFI_AP_CHANNEL":"6",
	"WIFI_AP_COUNTRY_CODE":"US",
	"WIFI_AP_MHZ":"20",
	"ENERGY_STAR_TIMEOUT":"240",
	"POWER_OFF_TIMEOUT_ENABLED":"OFF",
	"POWER_OFF_TIMEOUT":"240",
	"Vid":"0x29CC",
	"Pid":"0x101C",
	"Mfg":"Kodak Alaris Inc.",
	"Mdl":"S2080w Scanner",
	"DisplayName":"S2080w-64633610",
	"HostName":"S2080w-64633610",
	"SerialNum":"64633610",
	"PlatformVersion":"180506",
	"UserMode":"SINGLE",
	"ScannerModel":"S2080w",
	"OCPGraphicFormat":"PNG",
	"OCPGraphicWidth":"64",
```


Public

```
"OCPGraphicHeight":"64"}
}
```

Get Image of scanner

This enables access to a product specific PNG image, which can be scaled for use on your application.

Method: GET Function: **Image**

[GET] https://10.0.0.1/api/config/Image?Scanner

Params in body: none

Extra HTTP Header Values: none <- since there is no SessionId, this can be issued by anyone at any time.

Return Value:

Implementors Tips:

It is used within the web pages within the scanner and may be used within your application to show an image of the scanner.

Scale it as you deem fit. An example use of this is on the scanner's web pages.

Our use of this to display within our web page is to use:

```
<a href="https://www.kodakalaris.com/go/S2000w"><img title="" id="ScannerImageStatus"
src="/api/config/image?scanner" /></a>
```

With a CSS style to set the width, border, margin, and float, and a hover definition to scale the size.

Upload SSL/TLS Certificate

This call uploads an SSL/TLS certificate for secure HTTPS communication. Please reboot the scanner after applying.

Method: POST **Function**: Certificate **Query Param**: SERVER_CERT

[POST] https://10.0.0.1/api/config/Certificate?SERVER_CERT

Body:

ASCII PEM encoded SSL/TLS certificate. The certificate must contain the following items in order:

- RSA Private Key
- Scanner Certificate (CRT)
- Certificate Authority (CA) Certificate

Public

Return Values:

- “200 OK” for a valid certificate
- “400 Bad Request” for a malformed PEM file

Implementors Tips:

Follow these steps to prepare a PEM file for upload. All keys and certificates must be ASCII PEM encoded:

1. Generate a Certificate Signing Request (CSR) with RSA Private Key using this required info:
    a. Country = <Your Country>
    b. Organization = <Your organization name>
    c. Common Name = <The Scanner’s Display Name (including Domain Name, if applicable)>
2. Generate the scanner CRT by signing the CSR with a CA Certificate and its associated CA Private Key using this required configuration:
    a. Serial = <Unique certificate ID>
3. Concatenate the following ASCII PEM encoded data, in order, to create the SSL/TLS Certificate:
    a. RSA Private Key
    b. Scanner CRT
    c. CA Certificate
4. Use the SSL/TLS Certificate data as the body for the POST /api/config/Certificate?SERVER_CERT call.

Repeat these steps for each scanner that requires an SSL/TLS Certificate.

Raw HTTP Command Example:

```
POST https://10.234.10.175/api/config/Certificate?SERVER_CERT HTTP/1.1
Host: 10.234.10.175
Content-Length: 4236

-----BEGIN PRIVATE KEY-----
MIIEuwIBADANBgkqhkiG9w0BAQEFAASCBKUwggShAgEAAoIBAQDUjq24rB/rGPM2
1XaiSrgVi9kWZIxfsdXJ7cVS4z9JS8X1dW0vjxzP9yCGaclYuvf0bjFEoNkM1zri
KU1CzxpVfwhh15bxuroIUjnhTOXTq6DPOUdIJrPMjV92OSyD4r36+Ek2rbvdbwyD
8EL5NayAt37Ix/MDMbtUG1Dtywg5bCnK8nt4gRmeZG4KbS+p0t8uGMDsv/2IyvhZ
YQlkwumPWtISMHFmtyG96/TnnnfboNpBc42Hht87/EOxo2I2EZSWnXSvv6tKzF+D
zgCexYq02HGnWq1ZcKo/OtfJkotcmxkxk/2hftVPaBzFRWdvDvpOM89OYLZ5D2oW
LgFnda4BAgMBAAECgf9cjeCXOwaOig6y5xWr+OtaBK60UXeVbJpAPd3uQNrU7BRc
Kekvfhp4v3Guo+GeAVoE8aEe1l0CCJmwmefYxNVyq2jPasKdjbhlg6FM1eDoGLAe
4OdpCo2jmZY4Awe7nr/Tq+XDmyoAyNL4hBClnfmGaCdT4qB1oA6HNJasSfwnPPTt
soTlahksd3BUCK6tZwBAfrekEqzMSGBo9oxGgDB8WR0PtBggVY1kt9/3+eSTQ5gU
jhAeAfoWhAq/5Gvpbp3l5bDZcvROchka1J2eoKih1GBT8331jy/r4drekCsMSYHQ
UxcvHnONK/2nyoUzCMnxg8GlkU5OTCwRMsg/xn0CgYEA4eW+NdkWJkGoJ3lWIHr5
gvTpFxerXSaM5UoAZjnavGGNG8k8g7OQ/DWFYsHXw8mHWMhDX8sk9TxgBKRrEW+l
LuyMfIB49LxM3GHLspmVBOfv5AXgIB89buopaM1obNNm74T+cdLS7sOvOJ5G+dmX
IUBSFaR9yju7pPsqMoXlRY0CgYEA8OHaICcxjMXH0ajR4jv8vQZ68g1xzcgWwo7B
wzfPDynjz8E9UMcnI8xK6BfO0xVgHkAiY7Eex4Wn0DVedA5FavdqI5JHvOxgFyS5
hXFluXM+JMbPnjXvh8zWv1wrTxUDK82BSdNr3FST266LMpj6lg8pixtlGoC1Zr9z
aIz4a0UCgYEAOQLfkbdA9Ti+u3dbJ0ZhYrf0i3dNCS1IJEbqQ6u4JaElDNaPdVRK
C/IBbrToTLFE1/LD0/iEQ5wPQ3xVzJY+gkJAYloEkH1BqI+cB65u4VEw9wwwqcf9
8YgbtGqJ+ldH4MpvGyX0IRNSJtLb3QVk/lEjIf9GnD3UivwRuRmGX2kCgYA1THP8
K0/uaxDElksieM16ccWga1MkCkLm97lt3ZpmK4mpvYM4rWg+poRoiGx2jENBTkWN
s3IxCaTREA3/Ne0sNuR4THStG8xDMrAOE8JDX3l/n7HuutZuQOe/XcCDdg3Gkmkq
EarLDLfOovdP0cOYknBJxF0BmTsKKhB6ML8C1QKBgFF7J7SAK5kHvWnc9kY9yga6
q5CLwurZJjexDpDT2KYKcIRxmHT6TSrm65d/gGcfAwhGn33nQ6Brd6KTGyec99s+
LasWd5FWU35XYHY6lGaQDB/YKXtawtzPxq53KBqIIKGy2dWNExgttn+LyNp+wKg3
rCoOWTWEcx5sQrVcwX5r
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIDczCCAlugAwIBAgIEALxhTjANBgkqhkiG9w0BAQsFADA5MSAwHgYDVQQDDBdL
b2RhayBBbGFyaXMgU2Nhbm5lciBDQTEVMBMGA1UECgwMS29kYWsgQWxhcmlzMB4X
DTIzMDUxMTE4MzQwMFoXDTMzMDUwODE4MzQwMFowPTELMAkGA1UEBhMCVVMxFTAT
```


Public

BgNVBAoMDEtvZGFrIEFsYXJpczEXMBUGA1UEAwwOUzMwOTAtMTIzNDU2NzgwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDUjq24rB/rGPM21XaiSrgVi9kW
ZIxfsdXJ7cVS4z9JS8X1dW0vjxzP9yCGaclYuvf0bjFEoNkM1zriKU1CzxpVfWhh
15bxuroIUjnhTOXTq6DPOUdIJrPMjV92OSyD4r36+Ek2rbvdbwyD8EL5NayAt37I
x/MDMbtUG1Dtywg5bCnK8nt4gRmeZG4KbS+p0t8uGMDsv/2IyvhZYQlkwumPWtIS
MHFmtyG96/TnnnfboNpBc42Hht87/EOxo2I2EZSWnXSvv6tKzF+DzgCexYqO2HGn
Wq1ZcKo/OtfJkotcmxkxk/2hftVPaBzFRWdvDvpOM89OYLZ5D2oWLgFnda4BAgMB
AAGjfzB9MAsGA1UdDwQEAwIE8DATBgNVHSUEDDAKBggrBgEFBQcDATAZBgNVHREE
EjAQgg5TMzA5MC0xMjM0NTY3ODAdBgNVHQ4EFgQUrxNVQFR2fw89mdgItc2RYRTA
k60wHwYDVR0jBBgwFoAUCkm6hG6JJBoeEoOOH57idrbDjZEwDQYJKoZIhvcNAQEL
BQADggEBAI2F6K7IegGmS4skFm83/BOrFaxN8MAfxmShcGkEPjpqMYI6a1E6yQsM
kD8mb76d4EbP42RY8IYZlwMqDqiIAP7dxOGLCy+GxFxtt1zcyzDv7wgw8eWt+5Ky
TDXp8gW14QkYgau1YNoyysAXv9mkCniNNzaBhzBF3fdNKEwZYoCWu6cO2vIBa+ma
IzlxE9xsBi1XKur4dLCyiI4r7fETQk7yLaxY0Cah6TQdviziylXB12QhuccO/vsE
iFYS/dZL6sQRUVDVJJd8Gk5Pn7AYQAB5hTrAV7WaDzJT0K/+gSvtEmQpQENY/SeK
Q6+IZrlLkoHV7kWOuk3qjt2bhI4bnq8=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDUzCCAjugAwIBAgIUY/Z/zyYfJBAsmKNpw96DkP04Z+QwDQYJKoZIhvcNAQEL
BQAwOTEgMB4GA1UEAwwXS29kYWsgQWxhcmlzIFNjYW5uZXIgQ0ExFTATBgNVBAoM
DEtvZGFrIEFsYXJpczAeFw0yMzA1MTExODMxNTdaFw0zMzA1MDgxODMxNTdaMDkx
IDAeBgNVBAMMF0tvZGFrIEFsYXJpcyBTY2FubmVyIENBMRUwEwYDVQQKDAxLb2Rh
ayBBbGFyaXMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDOvm3Yo+VV
oHw/czZxw4pk3nHawbjSuJDiMZjcQeH3IHR7wv6j4u2b3VD2MW+VBm35agDfDvu0
gEgLx8ZsBBM2PNqKaCVNxr8TMwTjamZM6bFTKAW8+tZVL0JdgNMYIralzG0iZ7K8
p0UXbztcbi6lJRaoR45XvWyZMaUWNOb58BO40LpEMLh6vXhpR0z2+Onq7KCFU8F/
DP3+lrW4PqjHEAcnBcw3v8U/gLm/XJm2Y1rxv50eE6iLh0auYHkdV0vpTPkNl9pN
KTl4hlAtqF0VLtOFjXKH8P/S/diHzQN3nhGvaR4U9X6zhdqAEHKxRmCwKqadsLkc
JGbNbxhlcyjHAgMBAAGjUzBRMB0GA1UdDgQWBBQKSbqEbokkGh4Sg44fnuJ2tsON
kTAfBgNVHSMEGDAWgBQKSbqEbokkGh4Sg44fnuJ2tsONkTAPBgNVHRMBAf8EBTAD
AQH/MA0GCSqGSIb3DQEBCwUAA4IBAQBmohK7yhDPcrVI0L8KT1xEVlDV4MLL0q/x
YezlQgbD+uXzLL276nIj6AD9oYYtyA5avxPBVEGUolfPfSnR3bAuggSFWxByFH42
pB0aqInTcbv1w5fRfB7pEBT9NZUJccP5pQzruMay1vyjN3qt6N86CAo8rYHDTSfn
4tqpMJnMKEazyiZVXD9BeYr+Zi5M9nl7KG40MgoK8kKi5w56k0UgXOvYkDJtLzKw
RgHD0VtQi2o4FqG/nJtgDimzdxHIkCAvjaeeV1KKpRM8LMiQ2HJRFhOHeEVW+B6I
X7TuoulCWLJjx9upliqCH2r/+oZIpmHWloPjE3NkigY9TAGmaJms
-----END CERTIFICATE-----

Raw HTTP Response Example:

```
HTTP/1.1 200 OK
Content-Length: 0
Status: 200 Success
Date: Thu, 11 May 2023 14:33:58 GMT
Server: lighttpd/1.4.55
```


Public

SESSION API

See 'Fiddler's Composer Scratchpad ' near bottom of the document for examples of using the API using an alternate method of putting the SessionId onto the URL of the API. We support setting SessionId in the header, or the URL. Also note that the IP address can be a DNS name provided when we register our scanner on your corporate network with your DNS server, for example: https://s2080w-64633610.kodakalaris.net/api/session

[method] api/session/<function>

The Session Web API follows the CRUD design, Create, Retrieve, Update and Delete.

Method	Function	Description
POST		Session Create: starts a new session. Returns a SessionId or throws an HTTP error if something is wrong, see Error section.
GET		Session Status&Config: Read the root of this tree, obtaining both Status and Configuration settings. Learn how the job is progressing, and when to call GET Image.
GET	Status	Session Status: Read the status branch to obtain job status information.
GET	Configuration	Session Config: Read the configuration branch to obtain the job configuration settings.
DELETE		Session End: End the current session and delete all images for this session.
DELETE	Image/#	Delete a specific image and metadata from the scanner to free up memory for continuous scanning. Optional command! # is an integer from 1 to NumImagesScanned.
PUT		Change Config: Sets the state and passes configuration branch parameters that are read from the GET api/scanner/configuration. Only allowed when scanner state is "IN SESSION" or "DONE SCANNING".
POST	StartScan	Moves the scanner state to "SCANNING". Takes an optional parameter to set the scanner display username.
POST	StopScan	Moves the scanner state from "SCANNING" to "DONE SCANNING". Completes the currently scanning page. This is a graceful stop, which can be restarted with another StartScan.
GET	Image/#	Read JPG for Color, Grayscale image, and TIF for BW image, which is set via PUT configuration. # is an integer from 1 to NumImagesScanned, which is read from GET Status.
GET	Metadata/#	# is an integer from 1 to NumImagesScanned, which is read from GET Status. This obtains information about the image that has been scanned.

Session Create

Method: POST Function: **empty**

[POST] https://10.234.10.175/api/session

Params in body:

{"OCPUserName": "Jim Smith"} <- users name

Extra HTTP Header Values: none <- No SessionId exists until this command completes successfully.

Queries to start a new session and retrieve the Session ID. This session number must be used within 5 minutes or the session will timeout. Create session locks out all other PC's from driving a scan session.

Return Value:

Public

SessionId - string which identifies this session. This is a non-predictable GUID in string form. Save it as a string – this string is used in subsequent calls. If an API does not require this string, then that API can be called at any time.

Raw HTTP Command:

```
POST http://10.234.10.152/api/session HTTP/1.1
Host: 10.234.10.152
Content-Length: 31

{
"OCPUserName":"Jim Smith"
}
```

Raw HTTP Response:

```
HTTP/1.1 200 OK
X-Frame-Options: DENY
Content-Length: 26
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Fri, 08 Apr 2016 02:59:06 GMT
Server: lighttpd/1.4.35

{"SessionId":"1251877821"}
```

Implementors Tips:

The timeout is reset on activity within the session API. Your use of this session is to start a scan and poll Session Status during a scan job, and the status poll should occur within a frequency shorter than 5 minutes, typically once every second during scanning. This will keep the session from timing out and enable the scan engine to offload images from our internal DSP, which is an extremely fast image processing circuit. If you implement creating a session on start of your application, you should poll status every few minutes to keep it from timing out. When it times out you should call Session End (Delete).

$OCPUsername is passed in the body of the HTTPS, in JSON format.
This enables us to display a string on the OCP. For example: “Reserved for <username>”. You may want to allow the <username> string to be configurable for each user within your application. If this field is missing or empty, we default to a OCP Username of “Mobile User”

Note that there are restrictions that we must enforce for length on the OCP display, which should be followed.
Please read the Timeout section above.
Note that OCP, stands to Operator Control Panel.

If you issue TWO Create Sessions. The second Create Session will fail with a “423 Locked” error response. Error of 408 means the scanner is WAKING and you should wait and retry this. WAKING is an unspecified time but may be around 2 seconds.

```
HTTP/1.1 423 Locked
Status: 423 Locked. Someone else has an active session.
Transfer-Encoding: chunked
Date: Thu, 10 Sep 2015 19:33:07 GMT
Server: lighttpd/1.4.33
```

Session End

Method: DELETE Function: **empty**

[DELETE] https://10.0.0.1/api/session

This ends the active session, freeing up all internal resources previously created within this session. This will delete all images from the scanner.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Return Value:

Success returns 200 OK response and no data. This will only fail if the SessionId is invalid or missing.

Raw HTTP Command:

```
DELETE https://10.234.10.175/api/session HTTP/1.1
Host: 10.234.10.175
SessionId: 1234
```

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Length: 0
Status: 200 Success
Date: Fri, 11 Sep 2015 19:19:49 GMT
Server: lighttpd/1.4.33
```


Public

Session Status

Method: GET Function: **empty**

[GET] https://10.0.0.1/api/session?SessionId=<sessionId> This reads the root tree from the scanner, which includes both Status and Configuration information.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Error returns:

If the State goes from Scanning to Error, you should either DELETE the session to abort the job or monitor the status until the ErrorNum goes to 0 and PUT a new scan job to restart scanning of that job. We do not support continuation of a scan after clearing the error. DELETE and PUT of a new scan job will delete all stored images and restart from NumImagesScanned=0. For more information about ErrorNum see section ErrorNum Table.

Implementors Tips:

YOU MUST DO THIS COMMAND DURING SCANNING, and not more than once per second. This command is the heartbeat of the scanning API. Once you issue a StartScan command, the paper will start moving and internally we will run at full speed. We use a large internal DSP memory buffer to cache images, once that is full scanning will pause. Internally we transfer from that buffer to another larger system memory buffer, which enables us to cache around 64MB of image data. If either fills, we will pause. To reduce use of the larger buffer you would use the new DELETE api/session/image?<imageNum> command.

It might take a few minutes for Pause to occur, so no major rush on this, but your process should include polling of Status until the need to call this status.State changes to "Done Scanning". If you don't call Status within five minutes, the job will timeout, the scan job will be terminated and deleted. We need to know that you are still there, by issuing any of the Web API commands to reset our five-minute timer. Use the conf/diag.html web page to play with the API and understand how this works before writing code.

Oct 2018 update:

We've added NumImagesScanned. NumImagesStored has changed. BUT the NumImagesStored changes will not impact software you have previously written. NumImagesStored will only change if you never call delete image. If you never call delete image, NumImagesStored will equal NumImagesScanned. The scanner can store around 50 images of color compressed 300 DPI images before the image buffer is full. You can monitor this from the new Status.PercentAvailable to see if your use of Delete is needed.

Note that SessionId can be in the header or the URL. Also note that CORS may not pass the header information through, if you are calling the scanner across domains, and CORS is enabled in the scanner.

Key Variables within Status:

Status.State is read only parameter, which starts at "Idle", once you Post to api/session you have created a session and from that point you use the SessionId to access the Session. You will then be in "In Session" state. Once you perform a StartScan, this moves to "Scanning" state at which time "NumImagesScanned" becomes very important. "Done Scanning" or "Error" define the final states from which you can end the session via DELETE, or start another scan via

“api/session/startscan”. From capabilities you can see all of the possible settings:
"State":["Idle","In Session","Scanning","Done Scanning","Error"].
Status.NumImagesStored and Status.NumImagesScanned starts at 0 and increases as pages are scanned. In duplex mode, these numbers will increase by 2 for each page. As soon as NumImagesScanned changes, you can perform a GET api/Image command to obtain the scanned image. Reading of images can be done while still in “Scanning” state. You then read pages 1, 2, and wait for NumImagesScanned to increase beyond 2.

If you request an image number greater than NumImagesScanned, you will get an error. For example:
Status: 416 Parameter out of range. Image request greater than number of images available! image:1, numImagesScanned:0

Once the “NumImagesScanned” changes, you can get the image or metadata, and when the “State” changes to “Done Scanning”, the final image count will be provided via “NumImagesScanned”.

If you are scanning a PDF + BW images. The NumImagesScanned in duplex mode will move from 0, 3, 6, 9… Where every third page is the PDF, starting at page 1. In Simplex mode the NumImagesScanned will be 0,2,4,6… where every other page is a PDF, starting at page 1. You should read the metadata for each image to determine exactly what information is available for the image.

If you delete an image, NumImagesStored will decrement, but NumImagesScanned will remain the total number of images scanned. You should not use NumImagesStored to access your images if you are using delete image.
Status.PaperDetected accesses internal sensors to determine if the paper sensor is blocked.
Status.MemoryAvailable defines the percent of memory available for scanning. This will start at 100% and decrease as images are scanned. If you get to 0% your scan job will stop with an error. You will be able to read the scanned images, but you must end the session to continue scanning. You will not be able to download all of the images that were scanned. You should use delete image to avoid this.

The Status command also acts as a KEEP ALIVE for the session. There is a five-minute timeout built into the scanner. If you get status once every five minutes, the scanner will not drop your session. If are disconnected, the scanner goes back to IDLE state, will drop your session and report a 405 error informing you that the sessionId is no longer valid.

The scanner relies on this command to pump its state. You’ll notice that when you StartScan/StopScan then Put Configuration you will get an error. This is due to the StopScan being a message to transition to “Done Scanning” and finish the active page that is scanning. This will work successfully: StartScan/StopScan/Get Status/Put Configuration.

Public

Raw HTTP command:

```
GET https://10.234.10.175/api/session HTTP/1.1
Host: 10.234.10.175
SessionId: 1234
```

Success returns:

NumImagesStored: 0..?? <- Could be more than 50, dependent on amount of storage on scanner.

NumImagesScanned: 0..??<- Could be more than 50, dependent on amount of storage on scanner, and the number of images deleted while scanning.

State: Scanning	<- "Idle", "In Session", "Scanning", "Done Scanning", "Error"
DPI=200	<- same as defined in PUT
ScanSide=Duplex	<- same as defined in PUT
ColorMode=Color	<- same as defined in PUT
SkipBlankPages=0	<- 0= False, 1=True
PaperDetected=0	<- 0=No Paper in Scanner, 1=Paper in Scanner
ErrorNum: 0	<- Scanner Error num. NOT last HTTP error. Is a status of the embedded scanner.
LastError: errorString	<- Last Status string with some details on last HTTP error that occurred.

Raw HTTP Response:

```
HTTP/1.1 200 OK
X-Frame-Options: DENY
Content-Length: 843
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, X-Requested-With, Origin,
Content-Length
Status: 200 Success
Date: Tue, 23 Oct 2018 14:25:18 GMT
Server: lighttpd/1.4.49

{
    "Status": {
        "NumImagesScanned": 9,
        "NumImagesStored": 9,
        "State": "Done Scanning",
        "ErrorNum": 0,
        "LastError": "Status: 200 Success",
        "PaperDetected": "0",
        "PercentAvailable": 99
    },
    "Configuration": {
        "DPI": 200,
        "ScanSide": "Duplex",
        "ColorMode": "Color",
        "SkipBlankPages": 1,
        "SkipBlankPagesContent": 5,
        "AutoStart": 1,
        "ColorDropOut": "None",
        "ColorDropOutAggressiveness": 0,
        "OutputType": "Images",
        "ColorAutoBrightnessMode": "Automatic",
        "ColorBalanceMode": "Automatic",
        "ColorBalanceAggressiveness": 0,
        "ColorBalanceRed": 0,
        "ColorBalanceGreen": 0,
        "ColorBalanceBlue": 0,
        "ForegroundBoldnessMode": "None",
        "ForegroundBoldnessAggressiveness": 0,
        "BackgroundSmoothingMode": "None",
```


Public

```
        "BackgroundSmoothingAggressiveness": 0,
        "BinarizationMode": "iThresholding",
        "BinarizationContrast": 0,
        "MaxDocumentLength": 140,
        "MultifeedSensitivity": "None",
        "MultifeedResponse": "Stop",
        "PostScanRotation": "Automatic",
        "EdgeFill": "None",
        "ScanSource": "Automatic300",
        "ImageBorder": "Add",
        "TransportHandling": "BestStacking",
        "DocumentFeederTimeout": 2,
        "DocumentFeederTimeoutResponse": "Pause",
        "JpegQuality": "Best",
        "HoleFill": 1
    }
}
```


Public

Session Status – Status Only (Added in FW v1.07)

Method: GET Function: Status

[GET] https://10.0.0.1/api/session/status

```
HTTP/1.1 200 OK
X-Frame-Options: DENY
Content-Length: 149
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Tue, 23 Oct 2018 14:25:18 GMT
Server: lighttpd/1.4.49
```


```
"Status":{
        "NumImagesScanned":9,
        "NumImagesStored":9,
        "State":"Done Scanning",
        "ErrorNum":0,
        "LastError":"Status: 200 Success",
        "PaperDetected":"0",
        "PercentAvailable":99}
```


Implementors Tips:

The response to this query returns only the status branch of the Session Status tree.

Session Status – Configuration Only (Added in FW v1.07)

Method: GET Function: Configuration

[GET] https://10.0.0.1/api/session/configuration

```
Returns:
HTTP/1.1 200 OK
X-Frame-Options: DENY
Content-Length: 585
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Tue, 23 Oct 2018 14:25:18 GMT
Server: lighttpd/1.4.49

{
    "Configuration": {
        "DPI": 200,
        "ScanSide": "Duplex",
        "ColorMode": "Color",
        "SkipBlankPages": 0,
        "SkipBlankPagesContent": 5,
        "AutoStart": 1,
        "ColorDropOut": "None",
        "ColorDropOutAggressiveness": 0,
        "OutputType": "Images",
        "ColorAutoBrightnessMode": "Automatic",
        "ColorBalanceMode": "Automatic",
        "ColorBalanceAggressiveness": 0,
        "ColorBalanceRed": 0,
        "ColorBalanceGreen": 0,
        "ColorBalanceBlue": 0,
        "ForegroundBoldnessMode": "None",
        "ForegroundBoldnessAggressiveness": 0,
        "BackgroundSmoothingMode": "None",
        "BackgroundSmoothingAggressiveness": 0,
        "BinarizationMode": "iThresholding",
        "BinarizationContrast": 0,
        "MaxDocumentLength": 140,
        "MultifeedSensitivity": "None",
        "MultifeedResponse": "Stop",
        "PostScanRotation": "Automatic",
        "EdgeFill": "None",
        "ScanSource": "Elevator100",
        "ImageBorder": "None",
        "TransportHandling": "Normal",
        "DocumentFeederTimeout": 2,
        "DocumentFeederTimeoutResponse": "Stop",
        "JpegQuality": "Draft",
        "HoleFill": 0
    }
}
```

Implementors Tips:

The response to this query returns only the configuration branch of the Session Status tree. This is of the same format of the data written to the scanner on a PUT Configuration.

Public

Session Configuration

Method: PUT Function: **Configuration**

This command changes the state of the current active session or starts a new session with this state.

[PUT] https://10.0.0.1/api/session

Or

[PUT] https://10.0.0.1/api/session/configuration

Implementors Tips:

The format of the GET /configuration and GET api/scanner/capabilities (defaults section) is identical to the PUT /configuration command. You can read the current settings, or read the defaults and put those defaults back to the scanner to reset it to default.

Parameters are in JSON format and selected from the capabilities list. If a parameter is missing previous settings will be used. This command supports changing any one or more parameters.

Changes to the configuration become active when the state is set to "Scanning". A 405 response means you are not in either "Done Scanning" or "In Session" modes.
"Status: 405 Method not allowed. Invalid State."

You can load all settings from either GET /api/session with the appropriate sessionId, or via api/scanner/capabilities and use the default group to modify and write back via PUT /api/session. If the settings are valid, they will be accepted in any OutputType mode. So, changing a color setting in BW mode will not cause an error response.

SkipBlankPages – will cause the scanner to skip pages with less than the SkipBlankPagesContent value in percentage from 0 to 100%. This setting is not valid if the OutputType is set to SinglePageColorPDFPlus2TIFs and will result in a 405 error.

SkipBlankPagesContent – The content value to skip a page in percentage from 0 to 100%. This only applies when SkipBlankPages is enabled.

AutoStart – when set to 0 causes a scanner prompt to start the scan job. A value of 1 means to attempt to pull paper to immediately start scanning. If no paper is present, then prompt on the scanner to start scanning.

ColorDropOut – enables you to select a color to drop from BW images. See the list of colors supported in the api/scanner/capabilities command:
`"ColorDropOut":["None","Red","Green","Blue","Orange","OrangeAndRed","Predominant","Multiple"]`
ColorDropOut only has an impact on BW mode documents only. See capabilities to learn more about command dependencies.
ColorDropOut of Predominant or Multiple requires setting of ColorDropOutAggressiveness.

OutputType – supported types are Image and SinglePageColorPDFPlus2TIFS. See the Session Configuration rules for details on use of this parameter. Image will result in jpg images for color and gray, or tif for BW. If output type is

Public

SinglePageColorPDFPlus2TIFS the resulting output is PDF followed by one or two tif images, depending on the ScanSide setting (Simplex, Duplex). Note that scanning with ColorMode set to Color_BW, and OutputType set to SinglePageColorPDFPlus2TIFS in duplex mode will increase the “status.NumImagesScanned” by 3, whereas OutputType of Image will increase the “status.NumImagesScanned” by 4.

MaxDocumentLength – defines the maximum size of the document you would like to scan. This is an integer value in tenths of inch’s resolution. Leaving this at 140 (14.0 inches), will work for shorter documents. If you scan larger than 14 inches, without changing this, will result in a paper jam during scanning.

MultifeedSensitivity – depending on the sensitivity, the scanner will observe Multifeeds and will perform the MultifeedResponse when a Multifeed is detected.

MultifeedResponse – the action to perform when a MultiFeed is detected. A value of “Stop” will stop the job, and the value of “StopAndGenerate” will stop the job and generate the last image.

PostScanRotation – rotates the image X degrees after it has been scanned.

EdgeFill – fills in the gaps at the edge of a sheet to create a smooth clean image.

ScanSource – allows you to select where paper will be retrieved from. For example, elevator vs document feeder vs flatbed.

ImageBorder – adds or remove the image border from the output. When adding a border, the added thickness is 1/4th of an inch.

DocumentFeederTimeout – the amount of time before the feeder stops and performs the Document Feeder Timeout Response.

DocumentFeederTimeoutResponse – the response to perform when the Document Feeder Timeout is reached. A value of “Stop” will end the job. A value of “Pause” will pause the job and user interaction at the OCP is needed to continue the job. A value of “PauseAndWait” will pause the job but the feeder will automatically continue when paper is detected in the feeder.

TransportHandling – allows you to choose how delicately the paper comes out of the transport after the sheet has been scanned.

JpegQuality – allows you to choose the JPEG compression quality for the output image. This only applies when the Color mode is set to “Color” or “Gray” and the output type is “Images”

HoleFill – when enabled, holes from a hole puncher will be filled in to visually remove the holes from the output image.

Public

Color, Color_BW and Gray options:

Use /api/scanner/capabilities to learn the min/max/default of each of these settings.

Color Brightness – to enhance brightness of each image.

Foreground Boldness – use this option for documents or forms where you want the foreground (e.g. text, lines, etc.) to be more or less prominent.

Background Smoothing – use this option for documents or forms with a background color will help produce images with a more uniform background color. This option improves image quality and may reduce file size.

Color and Color_BW options, Not Gray:

Color Balance - **Automatic**: adjusts the white background of each document to pure white. This option compensates for the variations that occur between different weights and brands of paper. This is not recommended for use with Photographs.

BW and Color_BW options:

Binarization mode and contrast setting. **iThresholding**: the scanner analyzes each document to determine the optimal settings to produce the highest quality image. This option allows scanning of mixed documents with varying quality (i.e., faint text, shaded backgrounds, color backgrounds) and when scanning with consistent document sets.

Contrast – allows you to make an image sharper or softer. Decreasing this setting will make the image softer and reduce noise in the image. Increasing this setting will make the image clearer and make light information more visible.

To interact with all options, access /conf/diag.html prior to programming them.

GET on this same HTTPS will result in a response that reflects the state you have set, or the initial default state. The Get to retrieve just the configuration settings is via: http://<IP>/api/session/configuration?SessionId=1026744520 where the SessionId is from the Session Create.

Params in PUT body:

See capabilities for list of options, defaults, and range of settings:

Extra HTTP Header Values:

SessionId=$SessionId

Return Value:

Success returns standard 200 success response.

Error returns:

Error will return an HTTPS error. After sending Command=Scan and while in the Scanning state, a call to this command will result in an HTTPS error (405 Method Not Allowed). It is possible to start scan again, without losing the lock on the scanner. Note that sending the StartScan will delete any previously scanned images and Status “Number of Images Stored” will go to 0.

Raw HTTP Command:

PUT https://10.234.10.175/api/session HTTP/1.1
Host: 10.234.10.175
SessionId: 1234

Public

```
{
	"Configuration": {
		"DPI": 200,
		"ScanSide": "Duplex",
		"ColorMode": "Color",
		"SkipBlankPages": 0,
		"SkipBlankPagesContent": 5,
		"AutoStart": 0,
		"ColorDropOut": "None",
		"ColorDropOutAggressiveness": 0,
		"OutputType": "Images",
		"MaxDocumentLength": 140,
		"ColorAutoBrightnessMode": "Automatic",
		"ColorBalanceMode": "Automatic",
		"ColorBalanceAggressiveness": 0,
		"ColorBalanceRed": 0,
		"ColorBalanceGreen": 0,
		"ColorBalanceBlue": 0,
		"ForegroundBoldnessMode": "None",
		"ForegroundBoldnessAggressiveness": 0,
		"BackgroundSmoothingMode": "None",
		"BackgroundSmoothingAggressiveness": 0,
		"BinarizationMode": "iThresholding",
		"BinarizationContrast": 0,
		"MultifeedSensitivity": "Low",
		"MultifeedResponse": "Stop",
		"PostScanRotation": "Automatic",
		"EdgeFill": "None",
		"ScanSource": "DocumentFeeder",
		"ImageBorder": "None",
		"TransportHandling": "ImprovedStacking",
		"JpegQuality": "Best",
		"DocumentFeederTimeout": 0,
		"DocumentFeederTimeoutResponse": "Stop",
		"HoleFill": 0
	}
}
```


Raw HTTP Response:

HTTP/1.1 200 OK
Content-Length: 0
Status: 200 Success
Date: Wed, 03 Oct 2018 20:28:05 GMT
Server: lighttpd/1.4.49

Session Configuration rules

1. Use the api/scanner/capabilities to learn all configuration options that can be set via this PUT.
2. We do not support SkipBlankPages on (1) when in Duplex and OutputType of `SinglePageColorPDFPlus2TIFs`.
3. `SinglePageColorPDFPlus2TIFs` can only be set when ColorMode is set to Color_BW.

Public

4. Color_BW mode is also called “Dual Stream” and is supported for OutputType of both Image and `SinglePageColorPDFPlus2TIFs.`
5. `SinglePageColorPDFPlus2TIFS`, result in one PDF document that has either one (simplex) or two (duplex) raster images on separate pages within the single PDF. It will result in either one (simplex) or two (duplex) TIF images.
6. Color_BW mode provides two images per scan side, one Color, one monotone BW.
7. The scanning process is identical in Dual Stream scanning mode for either image or PDF, and for simplex or duplex. Simply continue to poll api/session/status once per second, when NumImagesScanned is incremented, download the one or more images. Obtain details on each group of images from the Get Metadata command.
8. When scanning in Color_BW (Dual Stream) with an OutputType of Image the images will be provided JPG, TIF, JPG, TIF for each duplex page, resulting in 4 images available for each physical page.
9. When scanning in Color_BW (Dual Stream) with an OutputType of `SinglePageColorPDFPlus2TIFs` the images will be provided PDF, TIF, TIF for each duplex page, resulting in 3 images available for each physical page.

Public

Session Recreate and Scan

StartScanning api can be sent prior to the DELETE (unlock) of the scan session. This enables you to perform multiple scans inside the scope of one job and enables you to design your application to retain ownership of the scanner for any duration you wish. You must obey the timeout rules associated with the Create and perform keep alive commands to retain that lock.

When Status:State=DONE_SCANNING, you can perform another StartScanning command. Performing a StartScanning will purge all image data from your previous scan and acts like a DELETE and POST in one step. The $SessionId is the value you retrieved when you did your initial Session Create. See Session Configuration for parameters.

Session Start Scan

Method: POST Function: StartScan

[POST] https://10.0.0.1/api/session/startscan

This starts a scan. After this command calls to Session Status will enable you to monitor the state of the scan and GetImage to retrieve images as they become available.

A 200 (success) means a scan has been initiated.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Raw HTTP Command:

POST https://10.234.10.175/api/session/StartScan HTTP/1.1
Host: 10.234.10.175
SessionId: 1234

Raw HTTP Response:

HTTP/1.1 200 OK
Content-Length: 0
Status: 200 Success
Date: Mon, 14 Sep 2015 21:05:32 GMT
Server: lighttpd/1.4.33

Session Stop Scan

Method: POST Function: StopScan

[POST] https://10.0.0.1/api/session/stopscan

This optional command will stop an active scan.

Implementors Tips:

The session remains active and must be closed (DELETE) or restarted (StartScan). You could use this command to stop scanning after the currently loaded sheet of paper is scanned. You can then access all scanned images up to status.NumImagesScanned. Then restart the scan without losing the session lock. The new scan (StartScan) will reset NumImagesScanned to 0. Treat this command as a mechanism to clean up memory in the scanner prior to rescanning, or as a non-destructive stop. Alternately use Delete to clean up memory without stopping.

Public

A 200 (success) means a scan has been stopped. If there is an error stopping a scan you will receive a: "`Status: 405 Method not allowed. Error stopping the scan.`". Note that it is a good idea to call GetStatus after stopping a scan to determine if you have additional pages to download. You are not required to download all pages. You should call Session End to abort the current scan, if you don't intend to download any more images.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Raw HTTP Command:

POST https://10.234.10.175/api/session/StopScan HTTP/1.1
Host: 10.234.10.175
SessionId: 1234

Raw HTTP Response:

HTTP/1.1 200 OK
Content-Length: 0
Status: 200 Success
Date: Mon, 14 Sep 2015 21:06:31 GMT
Server: lighttpd/1.4.33

Session Get Metadata

Method: GET Function: metadata

[GET] https://10.0.0.1/api/session/metadata/$ImageNum

HTTP Header:

SessionId=$SessionId

Response in body:

Information returned in JSON format:

```
ImageNum=1                  <- this will echo the same ImageNum in your request header
SheetNum=1
PaperSource=Rear  <- this is CameraType in NG language (Front, Rear, Both, All)
ImageSize=<Size in bytes>
ImageWidth=<in pixels>
ImageHeight=<in pixels>
XResolution=300     <- dpi
YResolution=300,    <- dpi
BitDepth=24,                <- 1,8,24
"EndOfSheet":0,
"ImageFormat":"PDF"}        <- TIFF, JPEG or PDF
```

Implementors Tips:

ImageNum is from 1 to status:NumImagesScanned .

The above example shows that the front side of page 1 was skipped. Your first image is Sheet #1 and back size. If no pages are skipped your first read will show Sheet #1, front. Sheet number increments per physical page. So, you could get I1 S1 Front, I2 S1 Back, I3 S2 Front, I4 S2 Back and so on.

Public

If you turn SkipBlankPages to True, you will not get the blank pages to download – we throw them away. If you turn SkipBlankPages to False, we will not flag any pages that are thought to be blank.

Note if you set SkipBlankPages to False, this function does not have to be called. Downloading the TIF/JPG will enable you to query those files for the ImageWidth,ImageHeight, ImageSize and the ImageNum, SheetNum and PaperSource is predictable. We do not support advanced algorithms in our product that enable you to scan multiple images on a single page, or chunk image data (YET). This function is here to enable future support of that, if needed.
EndOfSheet indicates the page you are referencing is the last image on this sheet.
ImageFormat is set to either “JPEG”, “TIFF” or “PDF”.

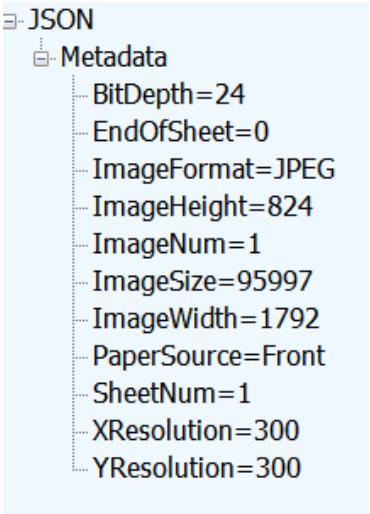
Raw HTTP Command:

GET https://10.234.10.175/api/session/metadata/3 HTTP/1.1
Host: 10.234.10.175
SessionId: 1234

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Length: 177
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Tue, 22 Sep 2015 14:16:44 GMT
Server: lighttpd/1.4.33

{"Metadata":
{
"ImageNum":1,
"SheetNum":1,
"PaperSource":"Front",
"ImageSize": 95997,
"ImageWidth": 1792,
"ImageHeight": 824,
"XResolution":300,
"YResolution":400,
"BitDepth":24,
"EndOfSheet":0,
"ImageFormat":"JPEG"} }
}
```



Public

Session Get Image

Method: GET Function: Image

[GET] https://10.0.0.1/api/session/image/$ImageNum

$ImageNum= is the image Number that you are querying and is from 1 to status:NumImagesScanned.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Response:

Image data in either JPEG, TIFF or PDF format.

This returns either a JPG for a grayscale or color image, a TIF for a 1 BPP bitonal image, or a PDF document for Color_BW mode.

Return Value:

Success returns an image of the full image size.

Raw HTTP Command:

```
GET https://10.234.10.175/api/session/image/10 HTTP/1.1
Host: 10.234.10.175
SessionId: 1234
```

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Transfer-Encoding: chunked
Date: Wed, 29 Jul 2015 20:58:15 GMT
Server: lighttpd/1.4.33
```

<image data>

Get Image Code Example:

```
// ipAddress is the IP of the scanner
// inLockId – is the sessionId you obtained on the Create Session (POST)
// inImageNum is the image from 1.. Status.NumImagesScanned from /api/session/status
function getImage(ipAddress, inLockId, inImageNum)
{
    var address = window.location.href;
    var protocol = (address.indexOf("https") == 0) ? "https://" : "http://";
    var portNumber = (protocol == "https://") ? 443: 80;
    var url = protocol + ipAddress + ":" + portNumber + "/api/session/";
    try {
        Image_window = window.open("", "Scanned Image", "location=0,status=1,width=700,height=800,", false);
        // if Image_window == undefined then pop up blocking is on – handle this
        Image_window.location.href = url + "Image/" + inImageNum + "?SessionId=" + inLockId;
    } catch (e) {
        debugger;
        DisplayStatus('Error loading image: ' + e.message, "Crimson");
        throw e;
    }
}
```


Public

Session Delete Image (Added to S2000w as of Firmware 180902)

Method: DELETE Function: Image

[GET] https://10.0.0.1/api/session/image/$ImageNum

$ImageNum= is the image Number that you are querying and is from 1 to status:NumImagesScanned.

Params in body: none

Extra HTTP Header Values:

SessionId=$SessionId

Return Value:

none

Raw HTTP Command:

```
DELETE https://10.234.10.175/api/session/image/10 HTTP/1.1
Host: 10.234.10.175
SessionId: 1234
```

Raw HTTP Response:

```
HTTP/1.1 200 OK
Content-Length: 0
X-Frame-Options: DENY
Date: Wed, 03 Oct 2018 20:35:53 GMT
Server: lighttpd/1.4.49
```

Implementors Tips:

This command is optional and removes one of the images from the scanners internal memory to free up more space for scanning. This is only needed if you intend to scan more than 50 images in one session. We will automatically delete all images at the end of a session.

It does not change any of the image numbers. Status.NumImagesScanned will not change once an image is deleted, but Status.NumImagesStored will decrement. If you delete image 1, then try to Get image 1, you will receive an:

Status: 404 unable to read this image, it may have been deleted.

Delete of image 1 will succeed (200), then another delete will fail (404).

Public

Session Get Meters (Added to S3000 as of Firmware v220203)

Method: GET Function: Diagnostics

[GET] https://10.0.0.1/api/session/meters

Response in body:

Information returned in JSON format:


```
{
	"Meters": {
		"PowerOn": "7379.4 Hours",
		"Transport": "13.4 Hours",
		"FrontLamp": "17.4 Hours",
		"RearLamp": "17.4 Hours",
		"FirstScan": "2017/9/26 13:45:00.000",
		"TotalPages": "78093"
	}
}
```


Implementors Tips:

This call returns the scanner's diagnostic and meter data.

This API call belongs to Session API but it does not require a session to be created.

The FirstScan datetime stamp is formatted as: "Year/Month/Day Hour:Minute:Second.Milisecond"

CORS support

A browser has security to protect accessing a Web API outside of its current domain. A device can be queried for support of Cross Origin Resource Sharing. We support CORS with Pre-Flight and reply with CORS headers in every response after support for CORS is turned on. Since this is a developer mode, there is no UI to configure CORS. There are at least three ways to turn this on/off. Note that this feature makes your scanner accessible from outside your current domain. The default configuration is OFF. Changes to this setting are sticky across reboots but will be turned off on a RESET.

To Turn on CORS and CORS Preflight support:

1) EasySetup, paste this into a browser:
   https://easysetup.kodakalaris.com/api/config/EasySetup?WEB_CORS_ENABLE=ON&EZSETUP_DESCRIPTION=Kodak%20Alaris%20CORS%20ON
   It generates a PDF, which when scanned in EasySetup mode will turn CORS support on.
2) Post: a body of WEB_CORS_ENABLE=ON to the URL: https://10.0.0.1/api/config/ApplyChanges which assumes your scanner IP address is 10.0.0.1. This will take immediate effect.

Example: https://10.0.0.1/api/config/ApplyChanges HTTP/1.1 Host: 10.0.0.1 WEB_CORS_ENABLE=ON

3) POST Using CURL you could paste into the console:
   curl -d WEB_CORS_ENABLE=ON https://<ip_address>/api/config/ApplyChanges -k --insecure

CORS Header response

The response from all of the Web API commands when CORS is enabled is:

```
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, X-Requested-With, Origin, Content-Length, SessionId
Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS
Access-Control-Allow-Origin: *
```

CORS Preflight

We support CORS Preflight. Preflight enables a system to request only the headers for every command to validate that the scanner supports cross domain sharing prior to requesting data.

Public

Error Codes – HTTP

All HTTP commands are parsed for completeness and accuracy. All responses follow the standard error response format, with HTML information and a header status field to aid in differentiating error responses.

For example:

Status: 400 Bad Request. We support Put, Post, Get, Delete.

Downloading the log from the scanner will provide much more detail on each call.

Error codes will be returned with HTTP error responses for all commands that do not suit our design, sequence format, or parameter list. Additional information will be returned in the response in JSON format if appropriate for that command. Listed below are the errors we generate via our API.

Error 500 means bug in server-side implementation. Typically returns like:

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/html
Content-Length: 369
Date: Fri, 11 Sep 2015 16:51:07 GMT
Server: lighttpd/1.4.33

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "https://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="https://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title>500 - Internal Server Error</title>
 </head>
 <body>
  <h1>500 - Internal Server Error</h1>
 </body>
</html>
```

All errors reported from the WebAPI will respond in this form:

```
HTTP/1.1 405 Method Not Allowed
Status: 405 Method not allowed. Session must be created first.
Transfer-Encoding: chunked
Date: Mon, 14 Sep 2015 15:00:01 GMT
Server: lighttpd/1.4.33
```

Report a bug on all Error 500 responses, this means Kodak Alaris bug. You should never see an error 500 response. Repeating the command MAY resolve the failure. It's also possible that on a 500, the command you issued took effect, and exiting from that command caused the failure.

We use 404 (Not Found) as our generic error message. The ErrorNum in the JSON block returned with the error string in the body. The HTTP errors and Status Error codes can be combined into one table on the application side.

HTTP/1.1 Responses	Description	JSON ErrorNum	JSON ErrorString
200	OK - Success	200	Success
400	Bad Request.		There are 56 different 400 responses with additional information to enable you to debug why your command is failing. • We support Put, Post, Get, Delete. • Mode unknown or invalid. Potential server-side code bug.

Public

			• Good command, exception processing command. • Command Failed. • Invalid request. Not allowed while someone is InSession. • Bad Request. Mode unknown or invalid. Potential server-side code bug. And some that you should never see, which only means you slipped a parameter past our checks which is invalid: • Bad scan parameters set at start of scan, possible timeout. • Bad camera (FRONT Camera) parameters set at start of scan. • Bad camera (REAR Camera) parameters set at start of scan. • Good command, exception processing command (1) • Good command, exception processing command (2)
403	Forbidden		• SessionId is empty. • SessionId is wrong. • Not the registered user in single user.
405	Method Not Allowed		Bad command or parameter, not following the documented rules. The reply from each 405 will include more details on exactly what is incorrect. There are at least 44 responses with additional information to help you understand what is not valid about your use of the API. • Session must first be created. • Scanning already active. • Fix error before scan can start. • No active scan. • This is not a valid command. • Session must first be created. • Method not allowed. Start of Scanning failed. • Invalid setting • Unable to load JSON (invalidly formatted JSON)
406	Not Acceptable		There are at least 28 different 406 responses that will inform you why the command parameters are not acceptable. • JSON Data missing. • JSON variable wrong. • JSON ScanSide wrong.

Public

			• JSON ColorMode wrong. • For example, setting DPI to 75 would return this error.
408	Request Timeout.		408 is returned in Create Session when the scanner is waking. Display waking, sleep for 1 or more seconds, and repeat the Create. See Timeout section for more details. 408 on Get Status means you timed out and took too long to call the scanner with a command. Five minutes will abort your job, leaving the scanner open to anyone for use. When this occurs the first poll of Get Status will change the state to Idle.
416	Parameter out of range		Image requested greater than number of images available.
420	Method Failure.		This may indicate a firmware bug, or misuse of an API.
423	Locked		Someone else has an active session. Try again.
500	Internal Server Error		Internal uncaught exception processing command. Firmware bug. Capture logs and report this to Kodak Alaris Support.
501	Not Implemented.		Not supported feature. Treat like a 405.
520	Unknown Error.		Bug in deciphering command or invalid command. Your code is calling an unsupported API.
522	Timeout		Deprecated – no longer returned by scanner. A 408 is the normal timeout response.

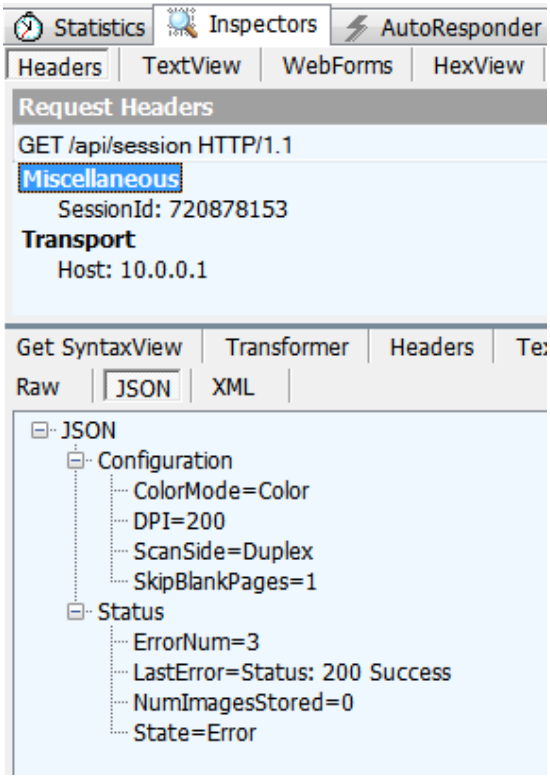
Error Codes – returned from Status

Internal scanner errors can result in the State changing to Error. In this case you would look at ErrorNum and the ErrorNum Table provided below.

Paper Jam:

"Status->State = Error and Status->ErrorNum=3 means Paper Jam.

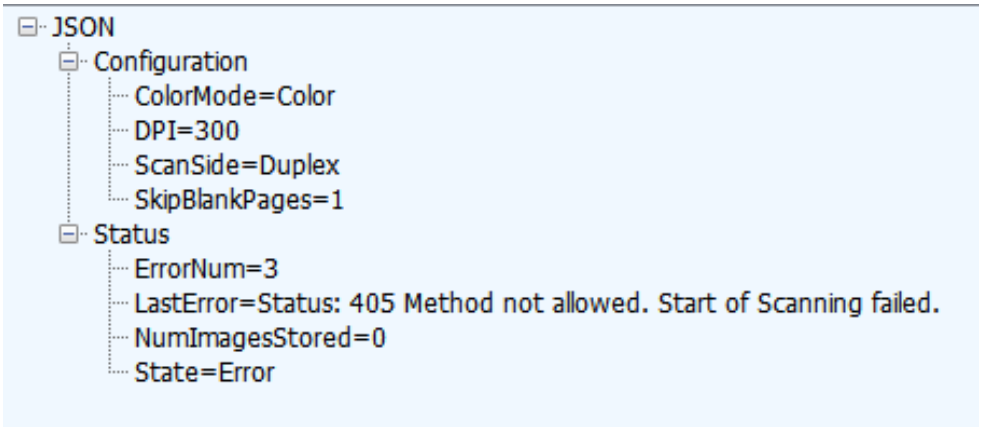
Public



The error occurs after you issue a StartScan command. Once a paper handling error, scanning is stopped. Polling of "Status" will not change the state after you clear the paper jam. You'll need to issue a StartScan command to refresh the state. During a paper jam the StartScan will return with an error 405 and the raw response looks like:

```
HTTP/1.1 405 Method Not Allowed
X-Frame-Options: DENY
Content-Length: 0
Status: 405 Method not allowed. Start of Scanning failed.
Date: Thu, 10 Mar 2016 19:05:37 GMT
Server: lighttpd/1.4.35
```

Notice in the header above, the "Status" field. This repeats the error but adds more information to aid in your determination of the cause of the error. It is also repeated on a subsequent "Status" command and put into LastError field, shown below.



Public

When the paper jam is cleared, StartScanning will return a response of 200 (OK), and GetStatus will show ErrorNum=0, “LastError=Status: 200 Success”, and State of “Scanning”.

ErrorNum Table

This table is taken from the 9.5 version of the host interface spec internal to Kodak Alaris Inc:

Status Description	**Status Value**	**Processing Information**
Success	00h	Returned when the command competed successfully.
No image ready	01h	The scanner is enabled and there are no images in the scanner’s image buffer. The host computer shall continue asking for images.
End of output	02h	Returned when the scanner is in the idle state and there are no more images in the scanner’s image buffer. The host computer shall stop asking for images. The host computer may use the information field for the number of images uploaded for the scan batch job. This allows the host to validate the number of images it has.
Paper jam	03h	Returned when the scanner detected a paper jam. Alert the operator of a paper jam.
Multifeed	04h	Returned when the scanner detected a multi-feed. Alert the operator of a multi-feed. (Only if the scanner was configured to end the job upon detection of the multi-feed)
Lamps warming	05h	The scanner’s lamps are warming up for scanning or calibration purposes. The command was not successful. * Data 1 – 2 (Data 1 is MSB) specify the Lamp Warm-up time (in seconds) needed for the lamps to warm up before executing the specified command.
Fail	06h	Returned when there is an unexpected error due to issues such as unrecognized command, invalid data fields, parameter check failure, or the scanner is in an invalid state for processing the command sent. Also returned when command execution has failed, which may occur during calibration or diagnostics. The operator should be notified to check the operator log for failure details.
Cover Open	07h	The scanner has detected that a cover is open and is failing the command.
Calibration Required	08h	The scanner has detected that a calibration is required.
White Patch Area Not Found	09h	Returned when the white patch was not detected.
Print Head Not Present	0Ah	Returned when the print head is detected as not present.

Public

Status Description	Status Value	Processing Information
Clean Imaging Guides and Re-feed	0Bh	Returned when the scanner detects that the imaging guides need to be cleaned and to tell the operator to re-feed documents
Feed Cartridge Not Present	0Ch	Returned when the feed cartridge is detected as not present.
ECDO Custom Lean Fail	0Dh	Returned when the custom ECDO learn algorithm fails.
Critical error	0Eh	The scanner is in the Critical Error state.
ECDO Custom Not Single	0Fh	The requested ECDO Custom Single was not a single, it was either not defined or defined as an ECDO Custom Multiple.
Unexpected Error	10h - FFh	These are unexpected status codes that the host may receive which could be used to support other interfaces. The host software will remap these received status codes to the Fail status code described above.

Timeouts

Two “timeouts” are implemented in the scanner to eliminate a chance of a deadlock:

1. The “Create Session” timeout is 5 minutes. Any **session** command will reset that timer. In other words, “api/Scanner/Status” will not reset the timeout; “api/Session/status” will.
    a. You would normally be polling the session status after you start a scan, waiting for the start button. An application that intends to implement a timeout while waiting for the start button to be pressed would be polling “session/status” and would implement its own application specific timeout.
    b. The default timeout is intended to trigger when an application connection fails: an application that starts a scan and stops polling status will timeout in 5 minutes.
    c. Your termination of scanning within a job can be initiated via either a DELETE or a StopScan. StopScan does not terminate the scan session. Delete does terminate the scan session.
    d. If you Create Session and do nothing for five minutes the scanner will release that session. Any access to the API using the old SessionId will result in either a 405 error (the scanner is not locked) or a 423 error (someone else has the scanner locked).
2. How to handle an error during Create Session
    a. Any error on a Create Session means you did not obtain the lock and you can try again.
    b. When the scanner is in low power and a Create Session is issued, the status “waking” is returned. This is done via a http 408 error response. The scanner will return the waking error for up to 12 seconds.
    c. Send the Create Session command again until the scanner returns a 200-success response. A one second delay in responding with this error is enforced by the web server to keep hosts from rapidly polling the scanner.

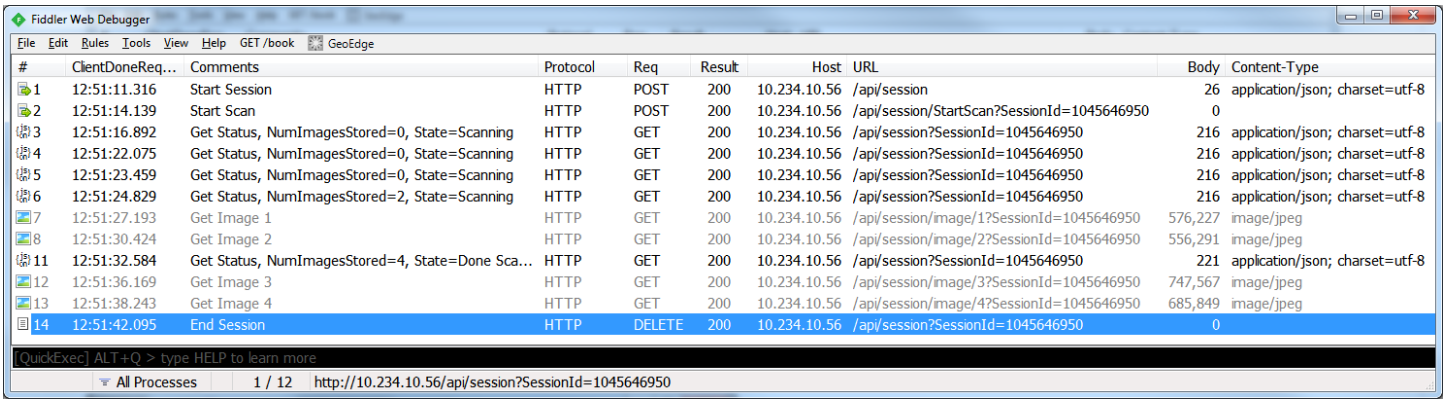
Public

More Details

Scan Example

1) Create Session (POST /api/session), error 408 means scanner is waking – repeat this until you get a 200.
2) Optional: Set parameters (PUT /api/session) with configuration parameters in the body (JSON format)
3) Start Scan (POST /api/session/StartScan)
4) Loop on Status (GET /api/session)
    a. If (Status->NumImagesAvailable is greater than the last image you have downloaded) then GET /api/session/Image/# where # is the image you want to read from 1 to Status->NumImagesAvailable.
    b. If (Status->State is not equal to "Scanning") then you have completed scanning either with an error or successfully. On success, complete downloading all images. You MUST call GET Status to update the state of the scanner.
5) Optional: Call Get Metadata for the image (GET /api/session/metadata/#)
    a. The only reason to call Get metadata is if you needed to know the image size prior to downloading it, or if you had Skip Blanks turned ON, and you needed to know the page number and side (Front/Back).
6) Optional: Loop back to Start Scan, note the images on the scanner will be deleted on Start Scan and NumImagesAvailable goes back to 0.
7) End Session (DELETE /api/session). Also, after five minutes the session will timeout from inactivity.

How this looks recording with Fiddler Web Debugger:



Fiddler Web Debugger

File Edit Rules Tools View Help GET /book GeoEdge

#	ClientDoneReq...	Comments	Protocol	Req	Result	Host	URL	Body	Content-Type
1	12:51:11.316	Start Session	HTTP	POST	200	10.234.10.56	/api/session	26	application/json; charset=utf-8
2	12:51:14.139	Start Scan	HTTP	POST	200	10.234.10.56	/api/session/StartScan?SessionId=1045646950	0	
3	12:51:16.892	Get Status, NumImagesStored=0, State=Scanning	HTTP	GET	200	10.234.10.56	/api/session?SessionId=1045646950	216	application/json; charset=utf-8
4	12:51:22.075	Get Status, NumImagesStored=0, State=Scanning	HTTP	GET	200	10.234.10.56	/api/session?SessionId=1045646950	216	application/json; charset=utf-8
5	12:51:23.459	Get Status, NumImagesStored=0, State=Scanning	HTTP	GET	200	10.234.10.56	/api/session?SessionId=1045646950	216	application/json; charset=utf-8
6	12:51:24.829	Get Status, NumImagesStored=2, State=Scanning	HTTP	GET	200	10.234.10.56	/api/session?SessionId=1045646950	216	application/json; charset=utf-8
7	12:51:27.193	Get Image 1	HTTP	GET	200	10.234.10.56	/api/session/image/1?SessionId=1045646950	576,227	image/jpeg
8	12:51:30.424	Get Image 2	HTTP	GET	200	10.234.10.56	/api/session/image/2?SessionId=1045646950	556,291	image/jpeg
11	12:51:32.584	Get Status, NumImagesStored=4, State=Done Sca...	HTTP	GET	200	10.234.10.56	/api/session?SessionId=1045646950	221	application/json; charset=utf-8
12	12:51:36.169	Get Image 3	HTTP	GET	200	10.234.10.56	/api/session/image/3?SessionId=1045646950	747,567	image/jpeg
13	12:51:38.243	Get Image 4	HTTP	GET	200	10.234.10.56	/api/session/image/4?SessionId=1045646950	685,849	image/jpeg
14	12:51:42.095	End Session	HTTP	DELETE	200	10.234.10.56	/api/session?SessionId=1045646950	0	

[QuickExec] ALT+Q > type HELP to learn more

All Processes 1 / 12 http://10.234.10.56/api/session?SessionId=1045646950

Public

Scan API: URL and JSON Response

The commands to do a full 300 DPI, duplex, color scan:

RESTful command	JSON Response
POST http://S2080w-64633633.local/api/session	{"SessionId": "1765739049"}
PUT http://S2080w-64633633.local/api/session?SessionId=1765739049 With JSON data of: {"Configuration": { "DPI": 300, "ScanSide": "Duplex", "ColorMode": "Color", "SkipBlankPages": "0" }}	
POST http:// S2080w-64633633.local/api/session/StartScan?SessionId=1765739049	

GET http:// S2080w-64633633.local/api/session?SessionId=1765739049	{
	"Status":{
	"NumImagesStored": "0",
	"NumImagesScanned": "0",
	"State": "Scanning",
	"ErrorNum": "0",
	"LastError": "Status: 200 Success",
	"PaperDetected": "1",
	},
	"Configuration":{
	"DPI": "300",
	"ScanSide": "Duplex",
	"ColorMode": "Color",
	"SkipBlankPages": "0"}
	}

Put paper in the scanner and press Scan button on the scanner	

GET http:// S2080w-64633633.local/api/session?SessionId=1765739049	{
	"Status":{
	"NumImagesStored": "4",
	"NumImagesScanned": "4",
	"State": "Done Scanning",
	"ErrorNum": "0",
	"LastError": "Status: 200 Success",
	"PaperDetected": "0",
	},
	"Configuration": {
	"DPI": "300",
	"ScanSide": "Duplex",
	"ColorMode": "Color",

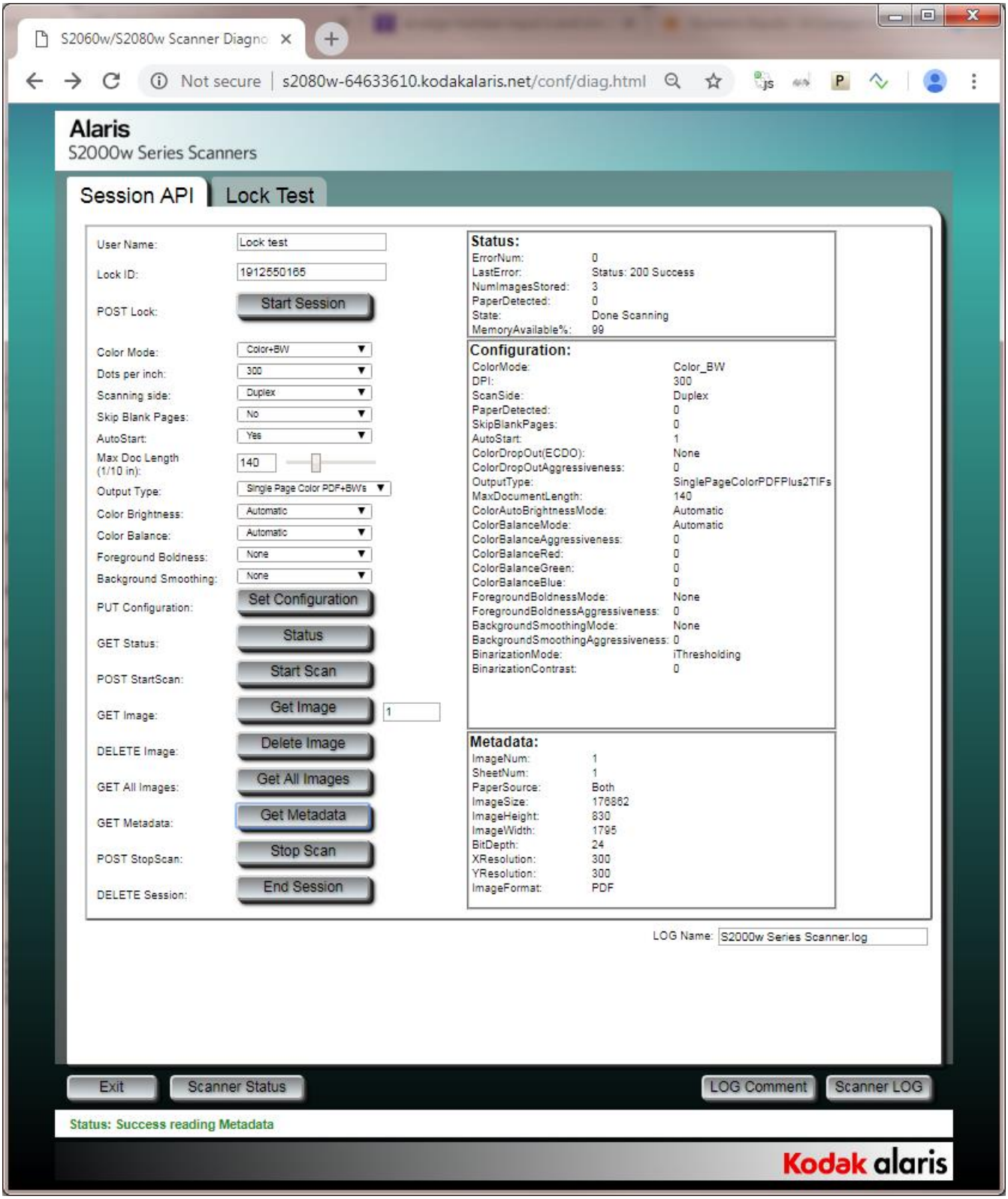
Public

	"SkipBlankPages": "0"} }
Keep issuing Get Status commands until the STATE changes to “Done Scanning”, or for “NumImagesScanned” to increase beyond the value you last read. I read 0 at start, and now shows 4, so I can get images 1 to 4. If when I read the Get Status – NumImageStored command again, it shows 5, or 6, I will read image 5 then 6. Once you have read all images and the State is at Done Scanning – you are done.	
GET http://S2080w-64633633.local/api/session/image/1?SessionId=1765739049	In this example an image/jpg of size 182769
GET http://S2080w-64633633.local/api/session/image/2?SessionId=1765739049	Second page
GET http://S2080w-64633633.local/api/session/image/3?SessionId=1765739049	Third page
GET http://S2080w-64633633.local/api/session/image/4?SessionId=1765739049	Fourth page
GET http://S2080w-64633633.local/api/session?SessionId=1765739049	Status shows I’m done so I can either start another scan or end the job with a DELETE
DELETE http://S2080w-64633633.local/api/session/StartScan?SessionId=1765739049	

Public

Programmers Page to play with the protocol

Diagnostic / Test page



Public

The conf/diag.html page is referenced in many locations in this document. This is a test page and is very helpful to implementors wanting to see the underlying JavaScript or html that is used to drive the API from a web page.

Since this is for implementers and breaks the Web API into its smallest parts.
Key to using this page requires understanding that a user would press the buttons in the order that an implementer would code within their application. There is not one button that does everything, although we provide example applications that do that. For example, the mobile application, or the python sample script called the Big Green button application (KaBgbScanningApp.py), both of which uses the Web API to drive the scanning functionality.

In this newest version of this page, we implemented some rules for showing the new scan configuration controls. Although only some controls are shown, all items in the PUT Configuration block are read and returned each time.

If you have not – please read about Session Status before proceeding.
In the pseudo code below **< >** means this step is optional. Also note that only pressing of the buttons initiates an API call to the scanner from the host browser. The drop-down selections and numeric fields are parameters for some of those calls.

Scan Example, pseudo code:

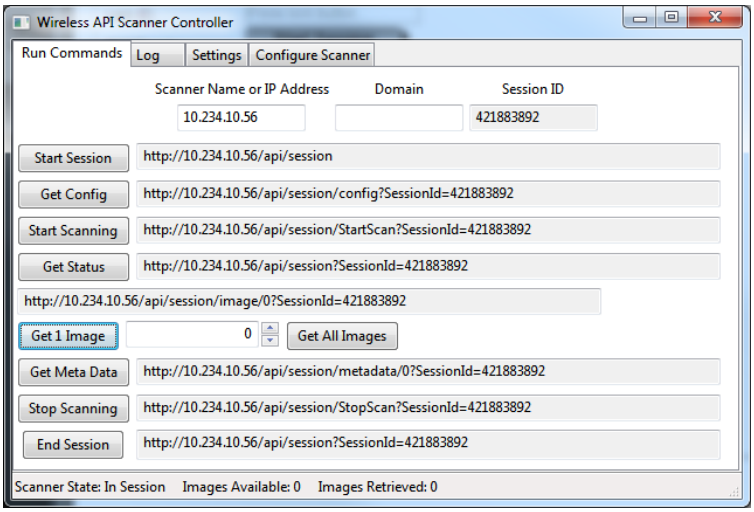
```
Press:
        Start Session
        Make the desired changes to the drop lists and Max Doc Length.
        Set Configuration
        <Status>                <- do this just to look at the return results from your Set Configuration to confirm everything is as you wish
        Start Scan
Loop start:
        Status                  <- you now must keep pressing this until State goes to Done Scanning
        If NumImagesScanned>0
                <Set the numeric control next to Get Image to 1>
                <Get Image>     <- display image 1.  Close or move aside when done.
                <Get All Images>  <- display all available images
                Sleep at least 1 second
        Loop back to "Loop start"   <- Repeat until top right Status: State goes to NOT "Scanning"

        Get image 1 then 2, then …
        Get All Images          <- this works well with color images, but Chrome/IE do not support reading TIF images when BW is selected
in OutputType.  Firefox does.
        Get Metadata            <- note that this works off of the numeric control next to Get Image to define what image metadata you
want.
        <Stop Scan>             <- this is NOT needed, unless you intend to stop scanning while paper is moving.
        End Session             <- This terminates the session and erases the "lock id".  Access to images from the previous session are no
longer possible.
```

You can experiment with these before writing code on http://<ip>/conf/diag.html

Public

Also use our Python example application, provided with the toolkit with source code:



Python scripts are available from Kodak Alaris technical support for Simple Scan, Scanner Controller, and a Big Green Button application.

Typical Sequence to use the API's

Discover zero or more scanners using Bonjour (See Discovery Protocols). Bonjour can take a few seconds to fully update. If you find a scanner, a few seconds later a second scanner can report it exists. You may want a dynamic list that updates as scanners show up, or a refresh button to update the display. If you use a static IP, DNS name or link local name to access your scanner, you can read all details of the name and information about that scanner from the Config API.

Prior to writing code - Learn about new capabilities

(GET) api/scanner/Capabilities to read the settings we support and their range of values. Outside and inside the scope of a session you can perform a GET Status and GET Capabilities. Update your display based upon the values in our configuration. Note that these values may change in the future to show a wider range of options. This command is designed to enable us to add more functionality and for you to learn of that functionality from the scanner.

Scanning

1. Create a Session
   (POST) api/session, store the "SessionId". If this succeeds you will retain ownership of that scanner until that session times out (5 mins), or you delete the session. At this point the user can safely put paper into the scanner and not be concerned that anyone but the application owning the session can scan it. If you close your application, or your application times out you must delete the session.

2. **Configure job**
   (PUT) api/session?SessionId=1234567890, optionally update the default parameters that configure the scan job or use the default settings. You can also perform a (GET) api/session/configuration to read all configuration parameters that you might want to modify.

Public

3. **Scan Start**
   (POST) api/session/StartScan?SessionId=1234567890 the scanner starts scanning, and we internally cache pages updating the Status. Access to those pages is secured via the SessionId returned from the Create Session.

4. **Poll Status**
   (GET) api/session/status?SessionId=1234567890 looking at NumImagesScanned and State. When this NumImagesScanned is greater than 0, you can read that number of images from the scanner. Store this number and continue monitoring for it to increment. NumImagesScanned is the index of the last image that has completed scanning.

5. **Read Images once NumImagesScanned increments**
   (GET) api/session/image/1?SessionId=1234567890
   Keep track of this number and keep polling until State changes to something other than "Scanning". As the NumImagesScanned increment – you can immediately read those pages.

6. **Completion options**:
   When State reaches "Done Scanning" you can either DELETE the session, to complete your session or start a new one via the Session Recreate and Scan. You can read Image Metadata during the scanning process to update your page. Now loop to #2 or #3 or continue scanning or continue to End Session. Optionally monitor the Status.PercentAvailable and use the delete image command to free up memory.

7. **End Session:**
   (DELETE) api/session?SessionId=1234567890 – ends your session deleting all internally stored image pages, unlocks access to the scanner for use by any other user.

To scan many pages

The scanner internally stores pages for a job. It holds them until you call StopScan or you end the session. So how would you scan a thousand pages given this limitation? StartScan – monitor status until you get 50 pages – StopScan, download all pages once the State turns to "Done Scanning", then call StartScan again. This sequence deletes all stored images and metadata freeing up space within the scanner. If done quickly it shouldn't slow down the scan mechanism. You also won't lose the persistent lock of the scanner, so no other host computers can start a scan job until you End Session. Note that when you perform a StartScan, the Status->NumImagesScanned variable will be reset to 0 (zero).

Security

No customer scan data is stored on permanently on the scanner. Once a job is complete or the connection times out, the whole job is erased. Connection to a scan job is secured via a session lock ID, which is only valid for that one session while scanning.

We recommend use of HTTPS. HTTPS provides an encrypted link to the scanner. The key within the scanner is private and secures your interaction with the scanner. The SessionId is a nonpredictive string which changes on every session. SessionId will be passed with every Session command except "Create", via one of two methods:

1. The header in a field called SessionId=2955999580
   Ex: GET http://i1150wn-58333625/api/session/image/1 HTTP/1.1
   Host: i1150wn-58333625
   SessionId:2955999580
2. The Request header.
   Ex: GET http://i1150wn-58333625/api/session/image/1?SessionId=2955999580 HTTP/1.1

The APIs support sending of the SessionId via the HTTP header or within the URL.
Ex: GET http://i1150wn-58333625/api/session/image/1?SessionId=2955999580 HTTP/1.1

Why use the Scanner API vs the Session API ?

The Session API drives scanning, and the Scanner API shows status and capabilities. The Scanner API is not needed to scan pages. It is useful to track changes to the API and to test status of the scanner without attempting to Create a session and to monitor the scanner status outside the scope of a scan job.
A GET http://S2080w-64633633.local/api/scanner/status
returns:


```
{"Status":{
        "Vid":"0x29cc",
        "Pid":"0x101C",
        "Mfg":"KODAK Alaris",
        "Mdl":"S2080w",
        "ScannerState":"Idle",
        "ErrorNum":0,
        "EstimatedMaxNumImages":50,
        "DisplayName":"S2080w-64633610",
        "HostName":"S2080w-64633610",
        "SerialNum":"64633610",
        "PlatformVersion":" 190103",
        "OCPUserName":"Vinnie"}}
```


This informs you that the scanner is IDLE. When the scanner is not “Idle” the OCPUserName tells you who is using the scanner.
The EstimatedMaxNumImages is fixed at 50 but is much more for B&W and different resolutions. Use the new Session API Status. PercentAvailable to track how full the scanner is becoming. Use the Session API Delete command to free up scanner internal memory.

How to determine the connection type from the IP

The CONFIG API is used for reading and changing connection settings.
Try:
GET http://S2080w-64633633.local/api/config/settings
This reads the current settings. What is unique about this is given the IP your scanner is configured for, you can resolve the connection type: Ethernet or Wireless. And if wireless one more command will enable you to resolve if it is Wireless infrastructure or ad-hoc.
http://S2080w-64633633.local/api/config/value?WIFI_MODE
the response looks like:


```
{
```


Public

"Variable":"WIFI_MODE",
"Result":"AP"
}
data.Result would be either:
"OFF", "AP" or "STA"
AP means Wireless (ad-hoc), referenced as Wireless Directly on the Scanner Web Pages.
STA means infrastructure aka Wireless.
When WIFI_MODE is "OFF" and ETH0_IP_MODE is not "OFF" then you are in Ethernet mode.

Discovery Protocols

Bonjour is an industry standard multicast discovery protocol available in Linux, Windows and native to Apple devices. It works very well at quickly finding all devices. We advertise our TCP protocol under _Scanner services, but the web protocol is available under _http HTTP service. In addition to obtaining the scanner's IP and port within the subnet, you can obtain information about those scanners from the TXT record.

TXT Record from Scanner for _scanner service:

txtvers=1
vid=29CC
pid=101C
mfg=Kodak Alaris Inc.
mdl=S2080w Scanner
HTTPS_Port=443
sn=64633612
path=/index.html
ty=KODAK S2080w Scanner Web Service

DNS Address

In addition to bonjour we register with any available DNS servers. So from within your company you may be able to connect to your scanner via HostName.<domain>. For example, I can resolve the IP address of the scanner via: S2080w-64633633.kodakalaris.net

Link-Local Address

You can also connect to your scanner via S2080w-64633633.local if you have a Link-Local name resolution service running on your PC. Installing Apple Bonjour service under Windows provides this service.

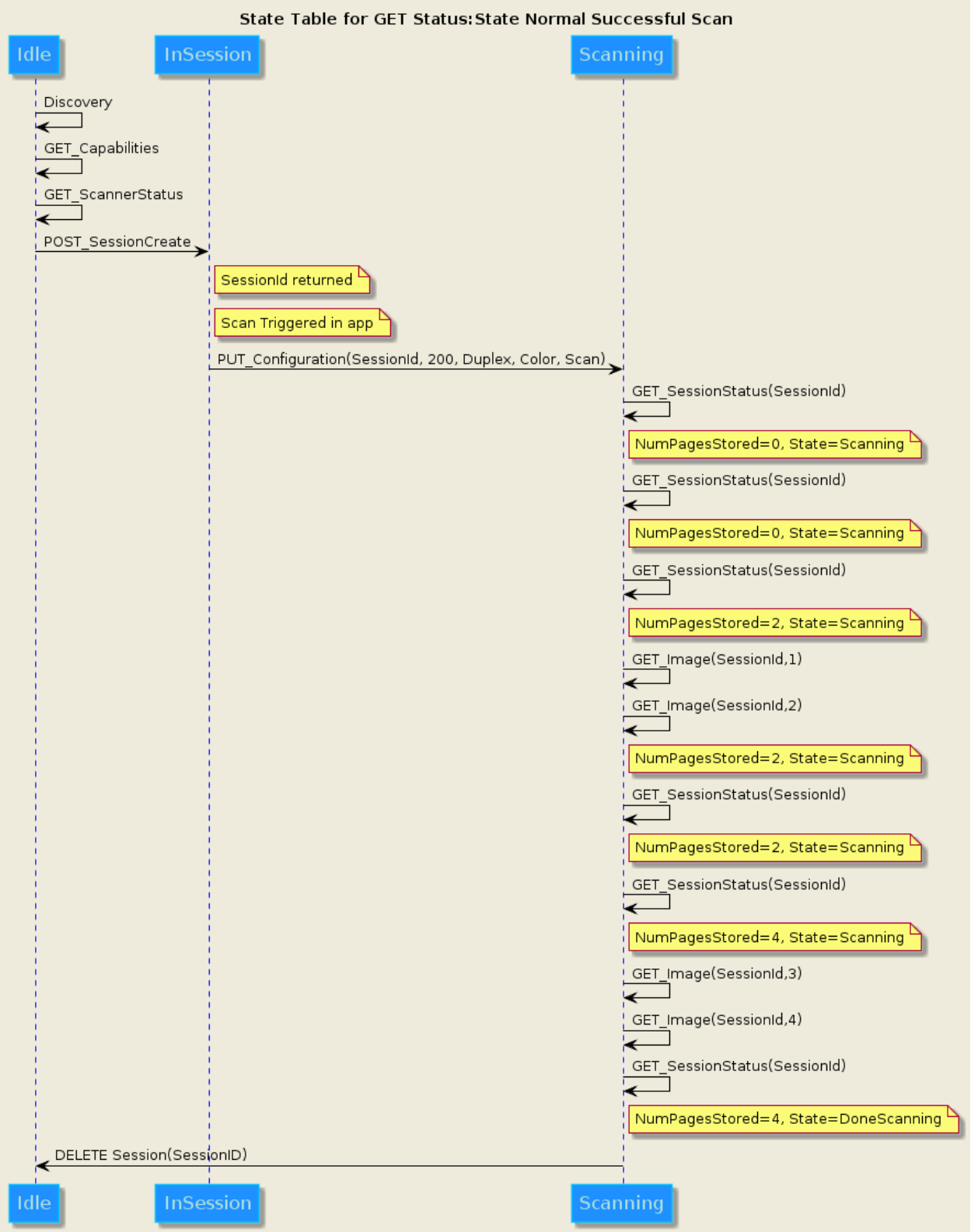
Static IP

With the scanner's IP, you can access the API's or web pages via that address. The previously described /api/scanner/status information may prove useful to validate the scanner is the one your desire.

Public

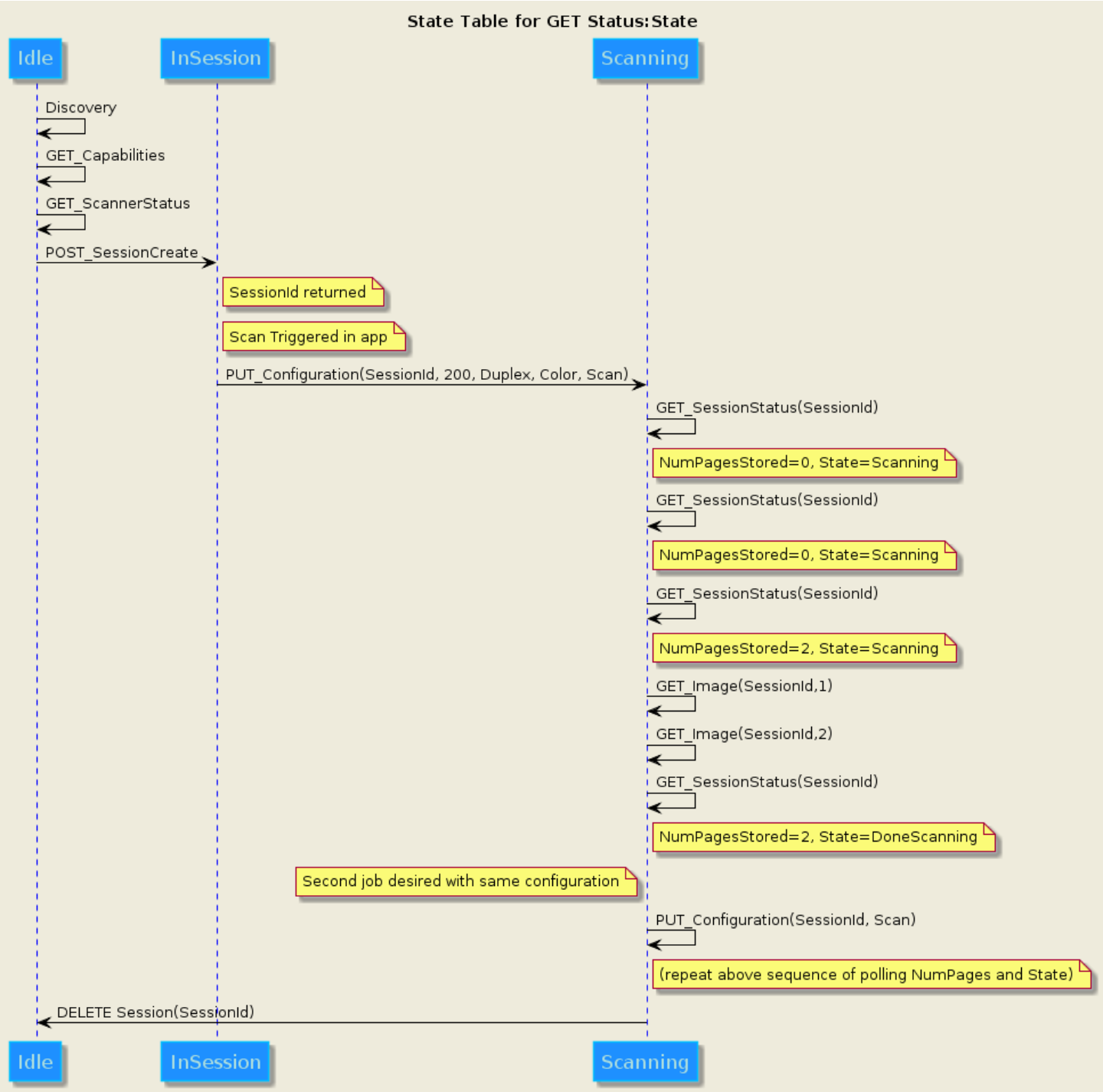
Diagram Scanning States

Normal Scan



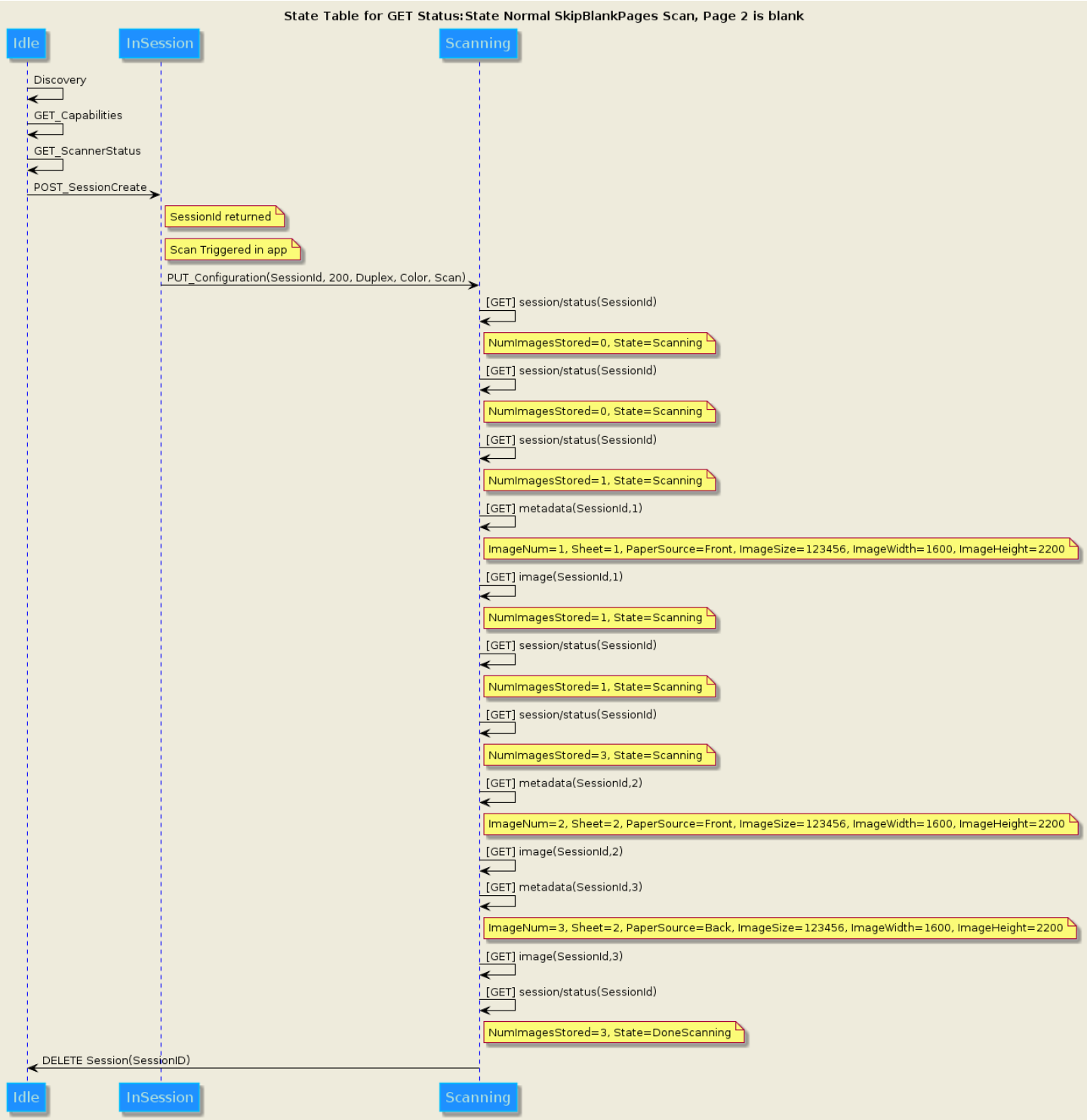
Public

Second Job Scan in same Session



Public

Skip Blank Pages – Back side of first sheet is blank, two sheets



Public

Fiddler capture of scan

This scan was done one command at a time to demonstrate the API. Errors were introduced to view the responses from the scanner.

Fiddler Web Debugger

File Edit Rules Tools View Help GET /book GeoEdge

#	ClientDoneRequest	Comments	Protocol	Req	Result	Host	URL	Body	Content-Type
115	12:21:18.927	GET Capabilities	HTTP	GET	200	kodak12...	/api/scanner/capabilities	221	application/json; charset=utf-8
116	12:21:22.784	GET Scanner Status	HTTP	GET	200	kodak12...	/api/scanner/status	156	application/json; charset=utf-8
117	12:21:29.529	Create Session, obtain SessionId	HTTP	POST	200	Kodak12...	/api/session	20	application/json; charset=utf-8
118	12:25:48.046	Get Session Status, Color, 200 dpi, Duplex, No Skip Blank Pages	HTTP	GET	200	Kodak12...	/api/session	170	application/json; charset=utf-8
119	12:28:05.680	Set Config: Color, 300 dpi, Duplex, Skip Blank Pages	HTTP	PUT	200	Kodak12...	/api/session	0	
120	12:28:37.919	Start Scan	HTTP	POST	200	Kodak12...	/api/session/StartScan	0	
121	12:28:56.708	Session Status: NumImages=0, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
122	12:29:06.095	Session Status: NumImages=2, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
123	12:30:42.285	Get Image 1	HTTP	GET	200	Kodak12...	/api/session/image/1	233,743	image/jpeg
124	12:30:58.081	Get Image 2	HTTP	GET	200	Kodak12...	/api/session/image/2	353,373	image/jpeg
125	12:31:25.728	Cause an error trying to get image 3	HTTP	GET	416	Kodak12...	/api/session/image/3	0	
126	12:31:58.440	Cause an error trying to start a new session	HTTP	POST	423	Kodak12...	/api/session	5	
127	12:32:06.962	Cause an error trying to read image 1, wrong SessionID	HTTP	GET	403	Kodak12...	/api/session/image/1	0	
128	12:33:21.364	Cause an error changing scan params mid scan	HTTP	PUT	405	Kodak12...	/api/session	0	
129	12:33:58.876	Emulate someone else asking for the scanner status	HTTP	GET	200	kodak12...	/api/scanner/status	160	application/json; charset=utf-8
130	12:34:29.519	Continue getting status: NumImages=2, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
131	12:34:36.227	Session Status: NumImages=4, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
132	12:35:29.560	Get Image 3	HTTP	GET	200	Kodak12...	/api/session/image/3	355,948	image/jpeg
133	12:35:38.058	Get Image 4	HTTP	GET	200	Kodak12...	/api/session/image/4	384,098	image/jpeg
134	12:35:46.989	Session Status: NumImages=4, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
135	12:35:52.354	Session Status: NumImages=6, Scanning	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
136	12:36:57.423	NumImages 8: Lets just jump to end, I think there are 50 Im...	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
137	12:36:57.639	Session Status: NumImages=10, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
138	12:36:57.830	Session Status: NumImages=12, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
139	12:36:58.027	Session Status: NumImages=14, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
140	12:36:58.181	Session Status: NumImages=16, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
141	12:36:58.349	Session Status: NumImages=18, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
142	12:36:58.525	Session Status: NumImages=20, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
143	12:36:58.702	Session Status: NumImages=22, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
144	12:36:58.869	Session Status: NumImages=24, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
145	12:36:59.046	Session Status: NumImages=26, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
146	12:36:59.205	Session Status: NumImages=28, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
147	12:36:59.381	Session Status: NumImages=30, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
148	12:36:59.549	Session Status: NumImages=32, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
149	12:36:59.733	Session Status: NumImages=34, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
150	12:36:59.901	Session Status: NumImages=36, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
151	12:37:00.068	Session Status: NumImages=38, Scanning	HTTP	GET	200	Kodak12...	/api/session	169	application/json; charset=utf-8
152	12:40:13.796	Get metadata: Image 1 - Sheet 1, Front Page	HTTP	GET	200	Kodak12...	/api/session/metadata/1	194	application/json; charset=utf-8
153	12:40:17.508	Get metadata: Image 2 - Sheet 2, Front Page	HTTP	GET	200	Kodak12...	/api/session/metadata/2	194	application/json; charset=utf-8
154	12:40:20.259	Get metadata: Image 3 - Sheet 3, Front Page	HTTP	GET	200	Kodak12...	/api/session/metadata/3	194	application/json; charset=utf-8
155	12:40:25.435	Get metadata: Image 4 - Sheet 4, Front Page	HTTP	GET	200	Kodak12...	/api/session/metadata/4	194	application/json; charset=utf-8
156	12:40:29.135	Get metadata: Image 5 - Sheet 4, Back Page	HTTP	GET	200	Kodak12...	/api/session/metadata/5	193	application/json; charset=utf-8
157	12:42:16.033	Get metadata: Image 6 - Sheet 5, Front Page	HTTP	GET	200	Kodak12...	/api/session/metadata/6	194	application/json; charset=utf-8
158	12:42:22.408	Get metadata: Image 7 - Sheet 5, Back Page	HTTP	GET	200	Kodak12...	/api/session/metadata/7	192	application/json; charset=utf-8
159	12:43:18.248	Cause error on Get Metadata - image number out of range	HTTP	GET	416	Kodak12...	/api/session/metadata/55	0	
160	12:44:03.549	Cause error on Get Metadata - Wrong Session ID	HTTP	GET	403	Kodak12...	/api/session/metadata/5	0	
161	12:44:37.590	Abort Scan	HTTP	POST	200	Kodak12...	/api/session/StopScan	0	
162	12:44:47.583	Get Session Status: Num Images 38, Aborted	HTTP	GET	200	Kodak12...	/api/session	168	application/json; charset=utf-8
163	12:46:02.418	I can still read images from the scanner - image 1	HTTP	GET	200	Kodak12...	/api/session/image/1	233,743	image/jpeg
164	12:46:10.219	Image 2	HTTP	GET	200	Kodak12...	/api/session/image/2	353,373	image/jpeg

Composer | Log | Filters | Timeline | Statistics | Inspectors | AutoResponder

Headers | TextView | WebForms | HexView | Auth | Cookies | Raw | JSON | XML

Request Headers [Raw] [Header Definitions]

GET /api/scanner/status HTTP/1.1

Transport

Host: kodak123456.kodakalaris.net

Get SyntaxView | Transformer | Headers | TextView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML

```
HTTP/1.1 200 OK
Content-Length: 156
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Fri, 09 Oct 2015 16:21:21 GMT
Server: lighttpd/1.4.33

{"Status":{
"Vid":"10700",
"Pid":"4114",
"Mfg":"KODAK Alaris",
"Mdl":"KODAK 1190wn",
"ScannerState":"Idle",
"ErrorNum":"0",
"EstimatedMaxNumImages":"50"}
}
```

Find... (press Ctrl+Enter to highlight all) | View in Notepad

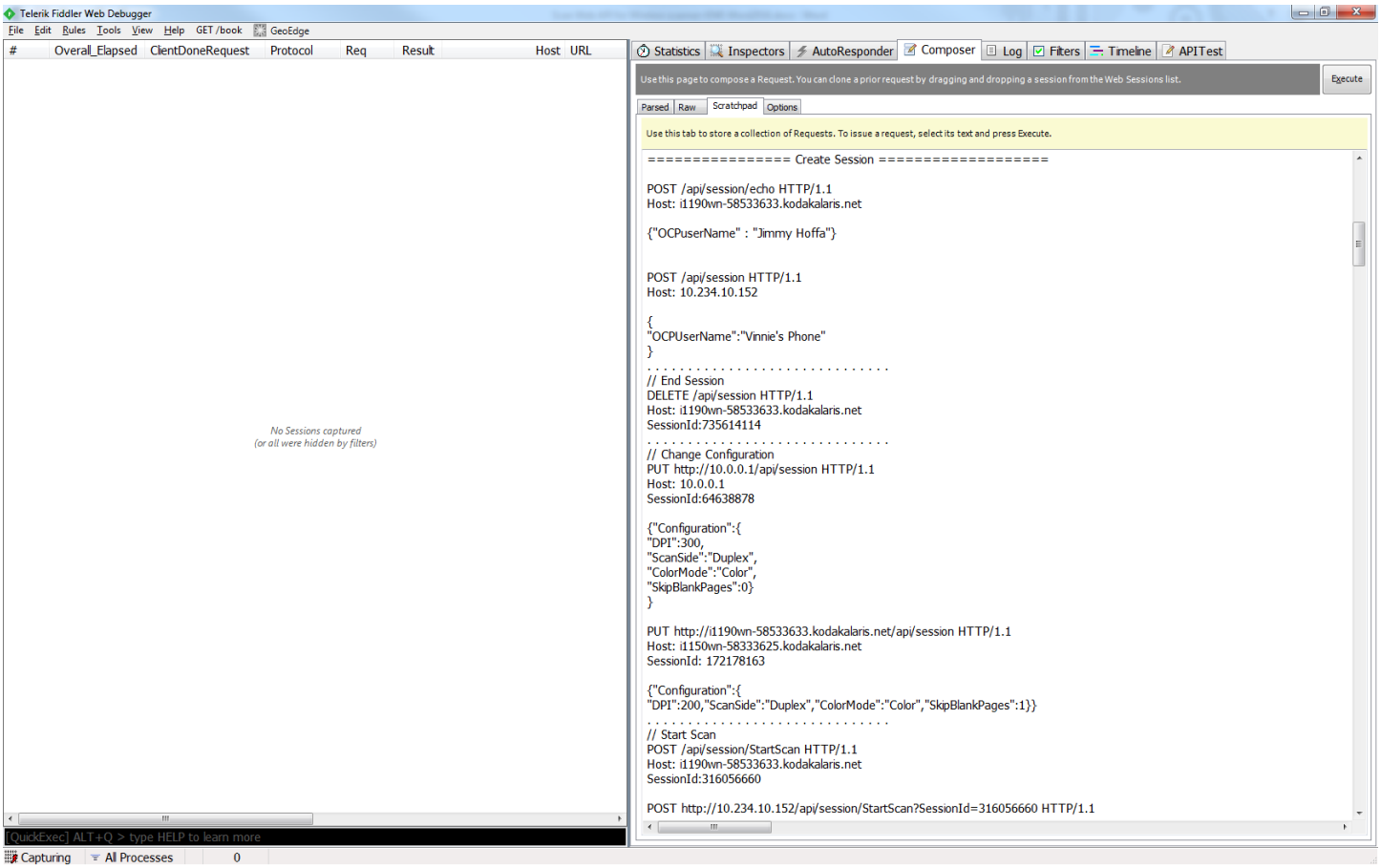
[QuickExec] ALT+Q > type HELP to learn more

All Processes | 1 / 55 | http://kodak123456.kodakalaris.net/api/scanner/status

Public

Fiddler's Composer Scratchpad

Paste this into the Fiddler Composer tab, Scratchpad.



The example below assumes the Create Session returned a SessionId of 583205357. You will need to edit that along with the IP address.

```
================ Create Session using the URL  ===================
POST /api/session HTTP/1.1
Host: 10.0.0.1

{
"OCPUserName":"Vinnie's Phone"
}
. . . . . .
// Change Configuration
PUT /api/session?SessionId=583205357 HTTP/1.1
Host: 10.0.0.1

{"Configuration":{
"DPI":300,
"ScanSide":"Duplex",
"ColorMode":"Color",
"SkipBlankPages":1}
}
. . . . . .
// Start Scan
POST http://10.0.0.1/api/session/StartScan?SessionId=583205357 HTTP/1.1
. . . . . .
// Stop Scan
POST http://10.0.0.1/api/session/StopScan?SessionId=583205357 HTTP/1.1
. . . . . .
// Get Status  - repeat until status = DONE_SCANNING
```


Public

```
GET http://10.0.0.1/api/session?SessionId=583205357 HTTP/1.1
. . . . . .
// Get Image 1
GET http://10.0.0.1/api/session/image/1?SessionId=583205357 HTTP/1.1
. . . . . .
// Get Metadata 1
GET http://10.0.0.1/api/session/metadata/1?SessionId=583205357 HTTP/1.1
. . . . . .
// End Session
DELETE http://10.0.0.1/api/session?SessionId=583205357 HTTP/1.1
```

Header version example, put session ID in header instead of URL.

```
. . . . . .
// Start Scan
POST http://10.0.0.1/api/session/StartScan HTTP/1.1
Host: 10.0.0.1
SessionId: 583205357
. . . . . .
PUT http:// 10.0.0.1/api/session HTTP/1.1
Host: 10.0.0.1
SessionId: 583205357

{"Configuration":{
"DPI":200,"ScanSide":"Duplex","ColorMode":"Color","SkipBlankPages":1}}
. . . . . .
// Stop Scan
POST http://10.0.0.1/api/session/StopScan HTTP/1.1
Host: 10.0.0.1
SessionId: 583205357
. . . . . .
// Get Image 1
GET http:// 10.0.0.1/api/session/image/1 HTTP/1.1
Host: 10.0.0.1
SessionId: 583205357
```


Public

!

Alaris

All trademarks and trade names used are the property of their respective holders.

The Kodak trademark and trade dress are used under license from Eastman Kodak Company.

© 2019 Kodak Alaris Inc.
TM/MC/MR: Alaris