Easy Start to Alaris RESTful Web API Scanner

Contents

To use the *Alaris* RESTful Web API Scanner 3
What is not covered in this document 3
Basics 3
Controllers 3
Session ID for Security 3
Session Timeout 4
Initial Connection 4
Connect to the scanner via Wi-Fi 4
Confirm your connection using a Web Browser 5
What happened, behind the scenes 5
Exercise: Use the Web API from a Chrome Extension 6
Loading a Chrome Extension/App 6
Connecting to the scanner 6
A full scanning sequence in the Chrome extension 7
Create the Session 8
Get Initial Status 10
Change Default Configuration for Scan 11
Start Scan 12
Get Status 13
Done Scanning 14
Metadata for Scanned Image 1 15
Read Image 1 16
End Session 17
Commands not used in the exercise 18
Scanner capabilities 18
StopScan 18
Diagnostic Application 19

To use the *Alaris* RESTful Web API Scanner

This document and the RESTful Scan Web API for Kodak Alaris Scanners are references for developers writing a programmatic interface to the Kodak RESTful Web API Scanners. A diagnostic application and Python scripts (with introductory explanation) are also provided.

The Web API provided with the Alaris RESTful Web API Scanners uses RESTful web services[1]. For developers who are not familiar with RESTful protocol, this document may be a better starting point than the RESTful Scan Web API for Kodak Alaris Scanners.

This document uses browser extensions rather than a specific programming language to show how to interact with our Scanner Web API. Only HTTP is used in these examples. The goal is to walk you through performing a full scan manually by the end of this document. You will use GET, PUT, POST, and DELETE and see the format of the information returned by the scanners in response to those commands.

What is not covered in this document

Programming languages, HTML, Java, JavaScript, or any language other than the RESTful API are not discussed. Discovery or finding the scanner on your network is not covered. Use of 10.0.0.1 assumes you are connecting to the scanner's "Wireless Directly" private isolated network, which is hosted from the scanner.

Basics

Controllers

This document refers to config, session, and scanner controllers. The Config API is intended to enable us to control the configuration of the wired and wireless connections to the scanner, and to access EasySetup PDF generation. The scanner controller is defined as a read only command language that enables anyone to query information about the scanner. The session controller is used to control a scan. These controllers are referenced from the http command. http://<IP>/api/<controller>/<function> breaks an http command into its parts. Some examples of the various controllers:

Controller	Function	HTTP API
Config	settings	http://10.0.0.1/api/config/settings
Scanner	Status	http://10.0.0.1/api/scanner/status
Session	Status	http://10.0.0.1/api/session/status

Session ID for Security

We need exclusive access to the scanner. We want security so no one can read our images while scanning. When we successfully Create a Session, we receive a SessionID. With every following call that pertains to

[1] REST: "Representational State Transfer": REST is the architectural style used for web development. Systems and sites designed using this style aim for fast performance, reliability, and the ability to scale.

RESTful systems typically communicate over Hypertext Transfer Protocol (HTTP) with the HTTP verbs (GET, POST, PUT, DELETE, etc.). A RESTful API can be used from any programming language: Visual Basic, C, C++, C#, PHP, HTML, JavaScript, iOS, Android, Windows Phone, and hundreds of foundations invented to enable access to APIs within a web browser. The Web API makes access from any operating system possible, even mobile OSs.

that session (e.g. start scanning, stop scanning, read images), that SessionID must be provided to the scanner. You can provide the SessionID in one of two ways:

1) append ?SessionId=12345678 to every API call to include the SessionID in the URL
2) put it in the header as SessionId:12345678.

Session Timeout

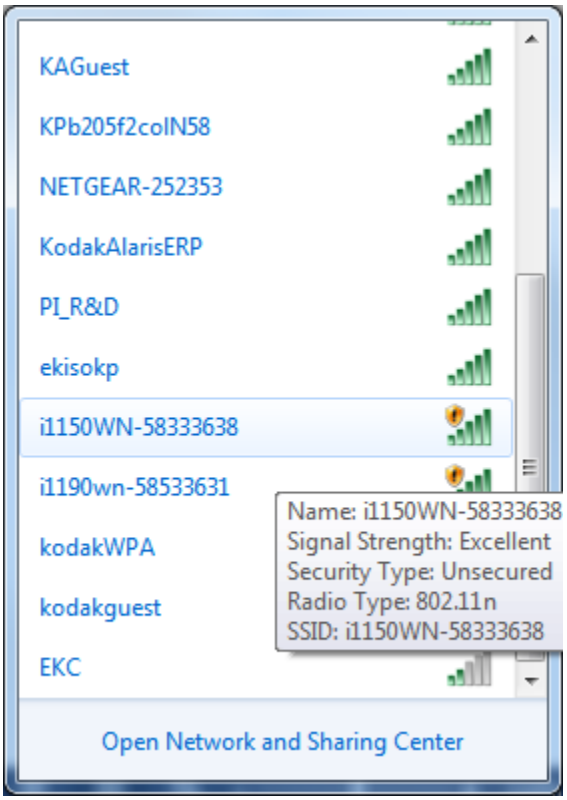
A session will time out after 5 minutes. To preserve the session, you must issue a call such as to start scanning, or read session status. If you wait more than five minutes, the session will time out, and you'll need to start over with Create Session to get a new SessionID.

Initial Connection

Connect to the scanner via Wi-Fi

Connect to your scanner wirelessly, by left clicking on the Wireless icon on your computer:

Select the private ad-hoc network for the wireless scanner:



Selecting the scanner ad hoc network will switch you off your internet connected network to the private network for the scanner. The "I" badge on the icon advises that you are not connected to the internet. You

can reconfigure your scanner to join your secure network, but for this exercise we will use the ad hoc wireless network provided by the scanner.

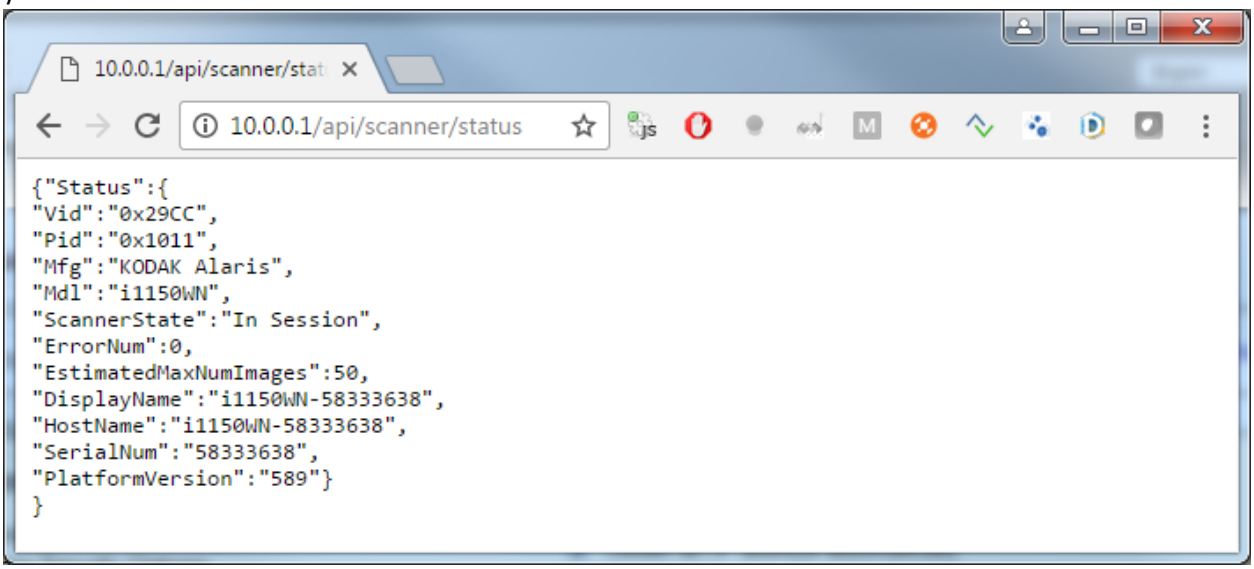
Confirm your connection using a Web Browser

Now confirm your connection to the scanner. Type into your browser:

http://10.0.0.1 to bring up the scanner's configuration web pages, or

http://10.0.0.1/api/scanner/status

This command returns a block of JSON data showing you some variables we use to drive an application on your PC.



You've just exercised a GET command that caused our Web API to gather information and return to your browser a JSON response.

What happened, behind the scenes

Even though you only ran a "scanner" controller with a simple query to the status function, the command via HTTP went over to our web server with the following information:

```
GET http://10.0.0.1/api/scanner/status HTTP/1.1
Host: 10.0.0.1
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,fr-FR;q=0.6,fr;q=0.4,zh-HK;q=0.2,zh;q=0.2,pt-
BR;q=0.2,pt;q=0.2
```

The web browser added much of the header settings, and the response came back looking like:

```
HTTP/1.1 200 OK
X-Frame-Options: DENY
Content-Length: 270
Content-Type: application/json; charset=utf-8
Status: 200 Success
Date: Wed, 04 May 2016 01:52:37 GMT
```

```
Server: lighttpd/1.4.35
```


```
{"Status":{
      "Vid":"0x29CC",
      "Pid":"0x101C",
      "Mfg":"KODAK Alaris",
      "Mdl":"S2080w",
      "ScannerState":"In Session",
      "ErrorNum":0,
      "EstimatedMaxNumImages":50,
      "DisplayName":"S2080w-64633610",
      "HostName":"S2080w-64633610",
      "SerialNum":"64633610",
      "PlatformVersion":"511",
      "OCPUserName":"..."}}
```


The response was composed by our web API and embedded web server. A developer can load the returned JSON data into a variable to query. Use of data.Status.HostName in jQuery language would return "s2080w-64633610". Use of data.Status.ScannerState would return “In Session”.

An application could use this information for a variety of purposes. It can validate the name of the scanner, the serial number, the firmware version, if the scanner is in use, and the username of the person using the scanner. The SCANNER API lets you monitor information about the scanner and the capabilities of the scanner and protocol.

Exercise: Use the Web API from a Chrome Extension

The above scanner/status API demonstrates a simple GET command using a web browser. The other commands can’t be entered into the browser. The following steps will exercise other commands.

Many tools can drive a RESTful API. Here we’ll use a Chrome extension called “Advanced REST Client”.

Loading a Chrome Extension/App

1. On your computer, open Chrome.
2. At the top right, click More ⋮ and then Settings.
3. At the left, click Extensions.
4. Scroll to the bottom and click “Get more extensions”.
5. In the “Search the store” box type REST
6. Now install “Advanced REST client”.
7. To find the “ARC” application after it is installed:
    a. Apps will show up in the Start->Chrome Apps location.
    b. Extensions load in the top right of the Chrome page.
    c. You can also type into the Chrome URL: chrome://apps/ to see icons of installed apps.

Connecting to the scanner

Connect to the scanner as shown in Connect to the scanner via Wi-Fi. Press the ‘i’ button on the scanner display and the IP address of the scanner, in this mode, would show 10.0.0.1. If it is different, change our use of 10.0.0.1 below to the IP address of your scanner.

Type into the URL:

http://10.0.0.1/api/scanner/status

or just click on the above link. If it doesn't work, then you're not connected to the same network as your scanner. Either connect to the same network or substitute the IP address of your scanner. Also verify that your desktop wireless icon is connected to the s2080w-64633610 network. Your host PC may automatically change connections due to it finding a preferred wireless network with a stronger signal strength.

My browser shows:


```
{"Status":{
      "Vid":"0x29CC",
      "Pid":"0x1012",
      "Mfg":"KODAK Alaris",
      "Mdl":"S2000w",
      "ScannerState":"In Session",
      "ErrorNum":0,
      "EstimatedMaxNumImages":50,
      "DisplayName":"s2080w-64633610",
      "HostName":"s2080w-64633610",
      "SerialNum":"64633610",
      "PlatformVersion":"511",
      "OCPUserName":"Vinnie Finn"}
}
```


A full scanning sequence in the Chrome extension

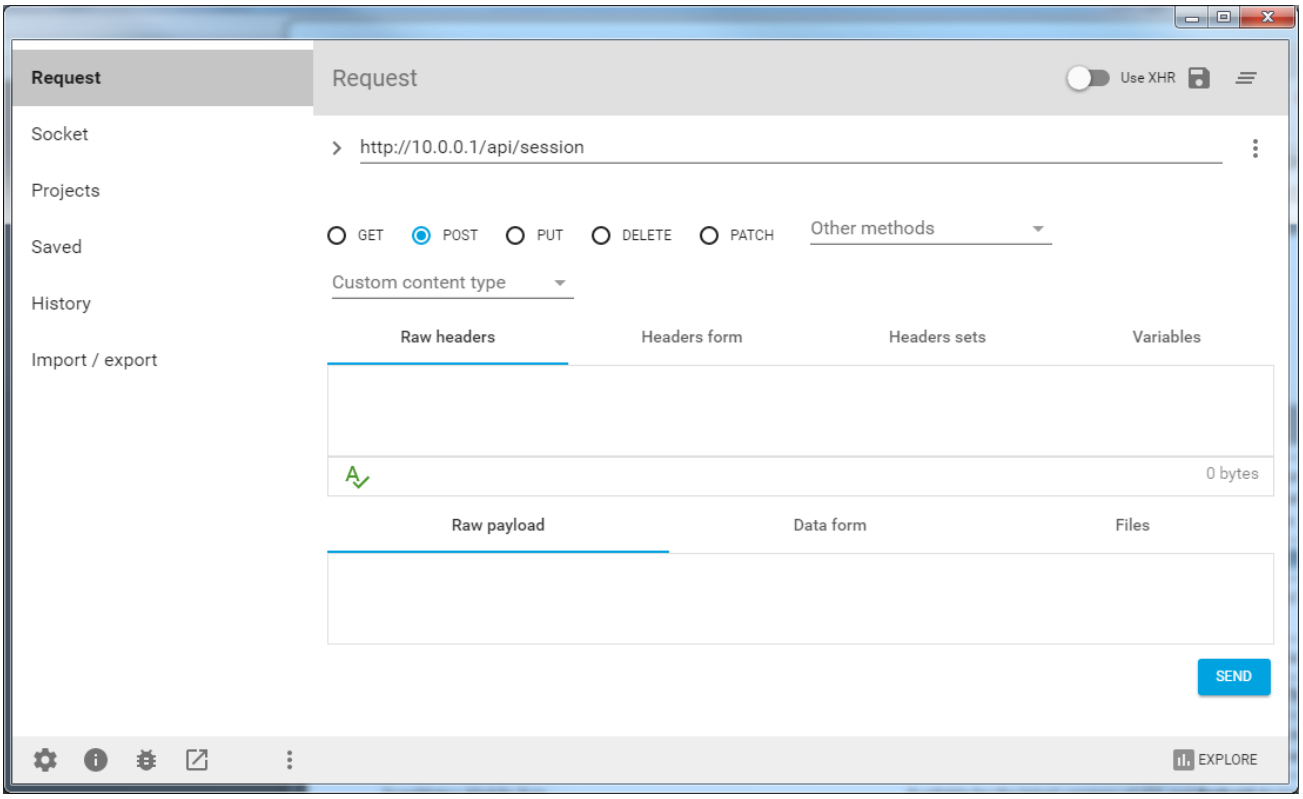
In this sequence we will do the following, in the Chrome extension:

1) Create a Session to get a SessionID
2) Change the configuration variables from the default settings
3) Start a scan
4) Monitor the progress (status)
5) Read images
6) End the session

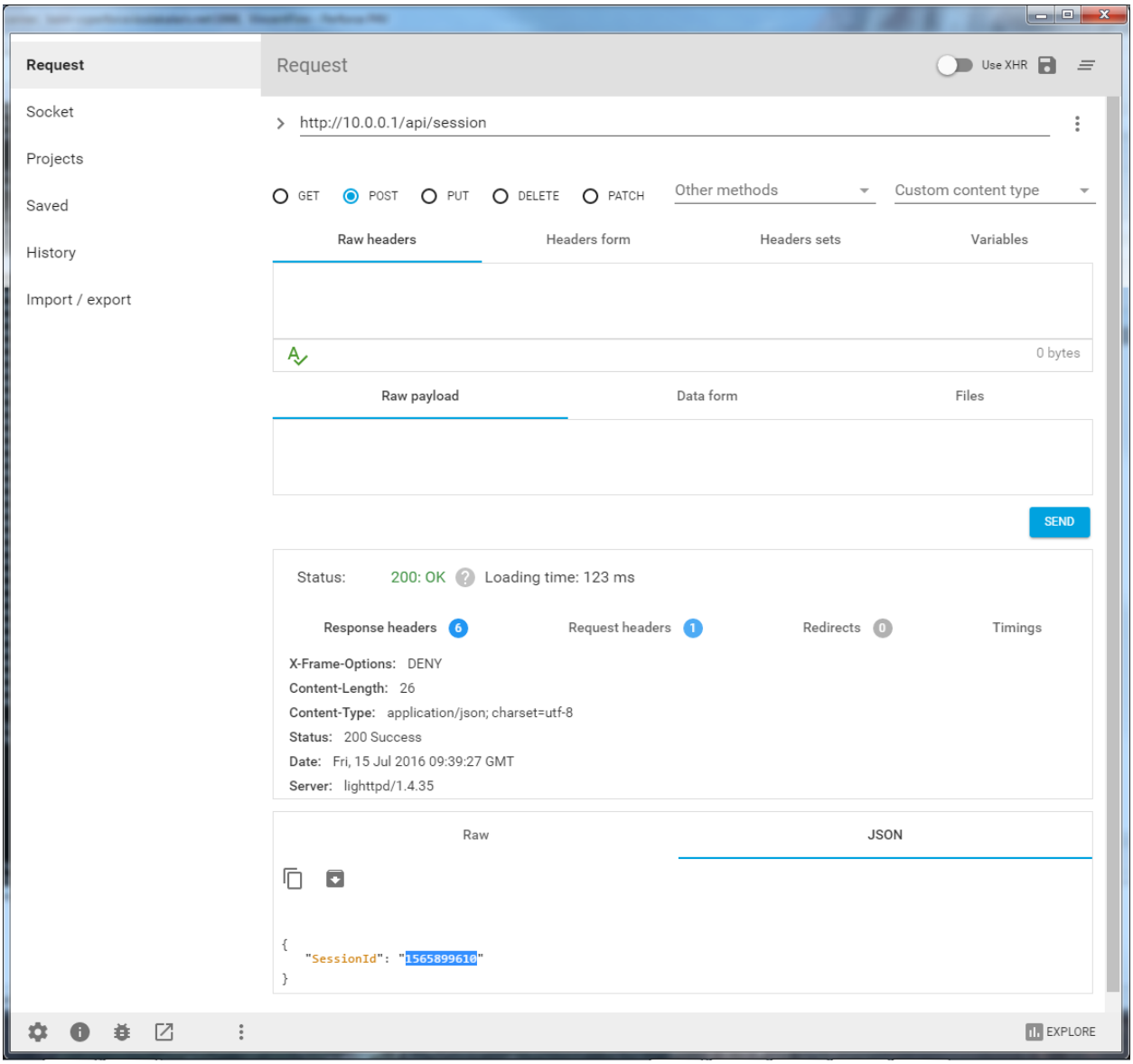
Create the Session

HTTP	API	Returns	Description

POST	http://10.0.0.1/api/session
	Optional parameter is OCPUserName to be sent in the Body in JSON format. For example:
	{
	"OCPUserName":"Vinnie's Phone"
	}
		SessionId:12345678	408 error means the scanner is waking.
			
			423 error means that someone else is busy scanning.
			
			200 means success.



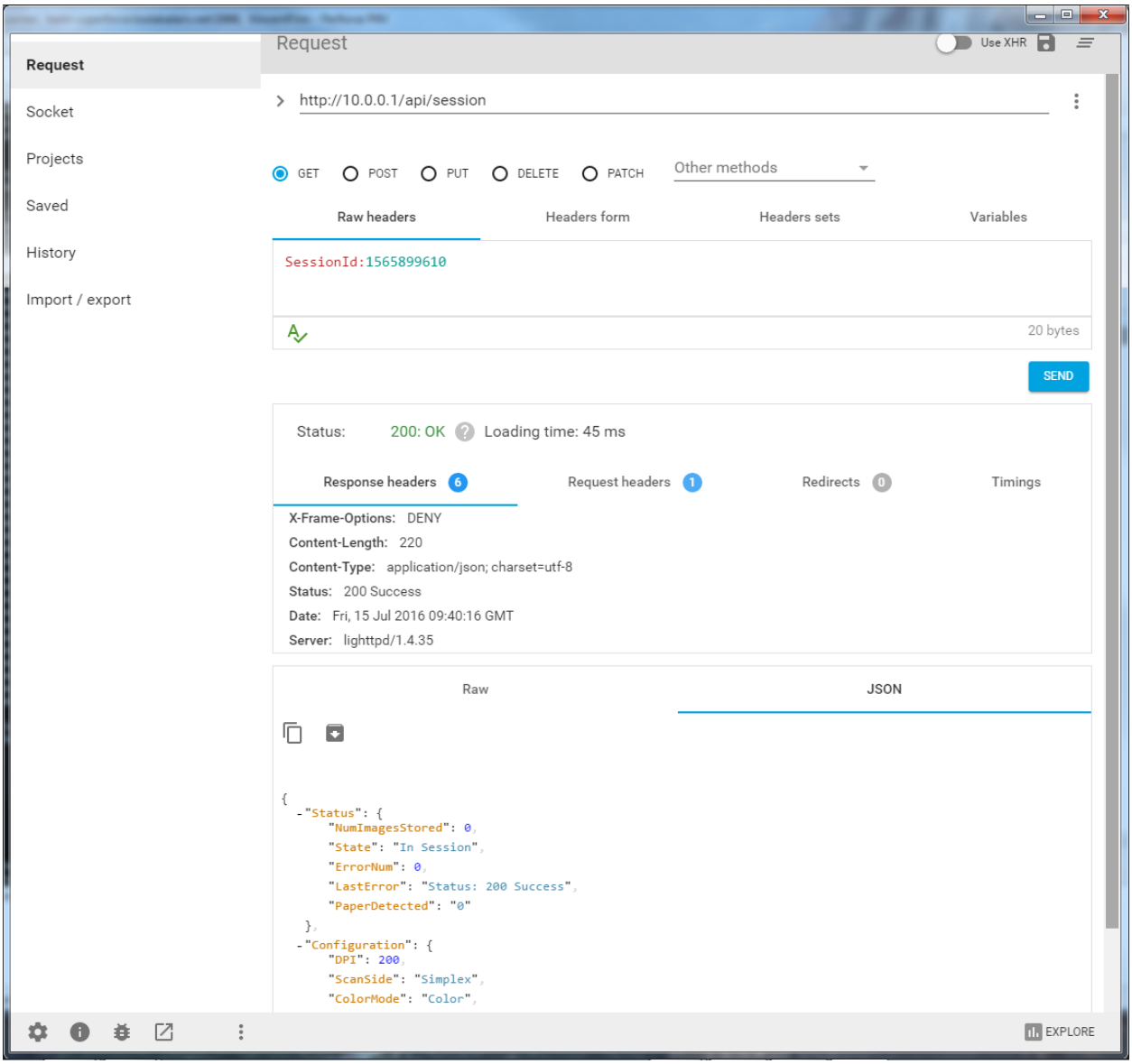
Press SEND. You may receive a 408 error code. Error code 408 is returned in Create Session when the scanner is waking. Repeat the POST command above until the response changes to a 200. Note: the scanner enforces a one second delay on a 408 error to keep it from being polled at a high frequency.



Above we see the response Status: 200: OK and a SessionID, highlighted in the screen shot above, in JSON form. The scanner OCP updates. Note we didn't set the OCPUserName, which is optional. Copy the session ID you receive, and paste into Raw Headers field, as shown in the next screen shot (and thereafter), for all the following commands.

Get Initial Status

HTTP	API	Returns	Description
GET	http://10.0.0.1/api/session	Status and Configuration	We support variants of this command that allow you access to only Status OR Configuration. Accessing the root gets both.

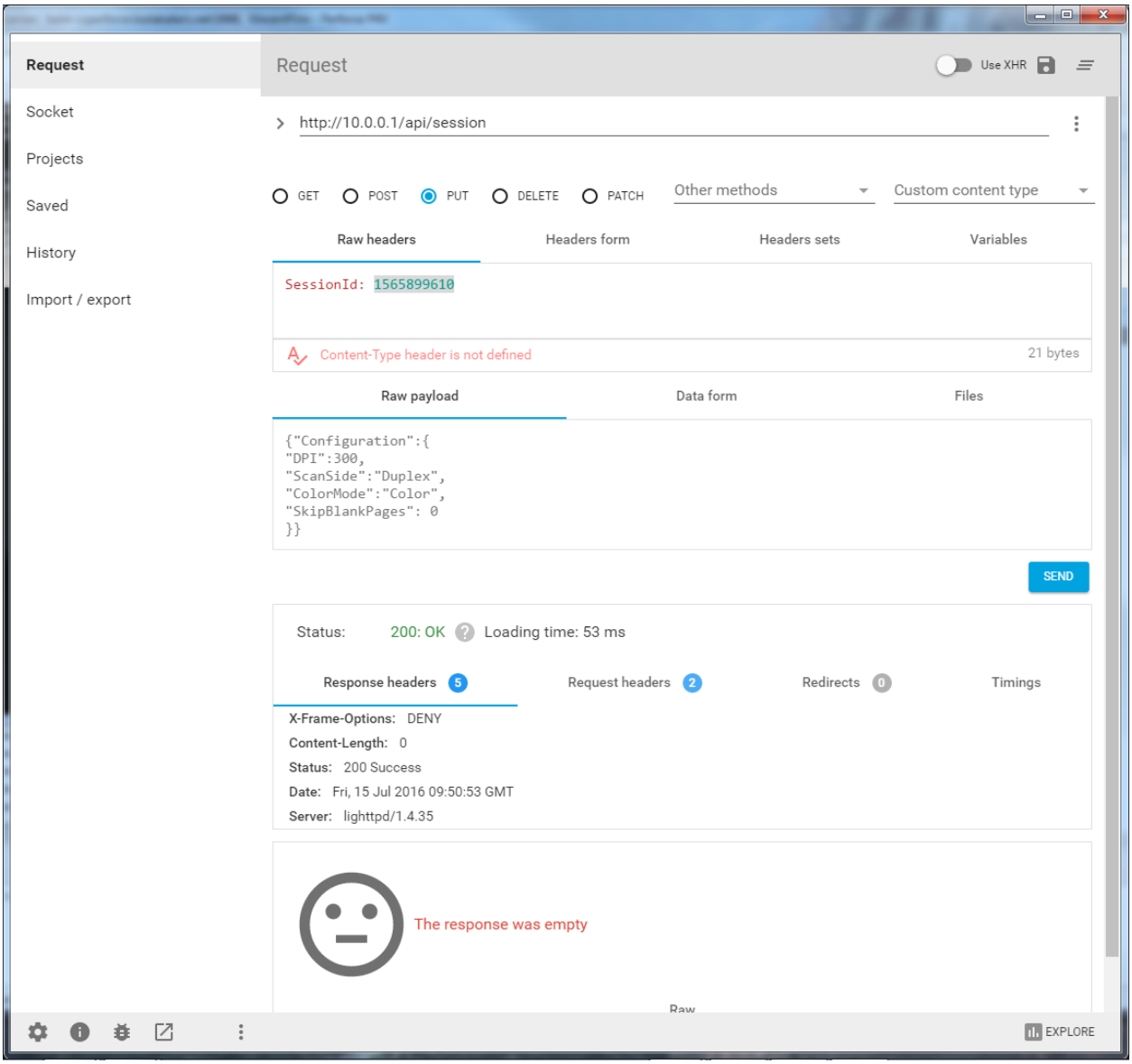


Returned information shows the state is in session and no paper is detected, and the default configuration for the scan job.

Change Default Configuration for Scan

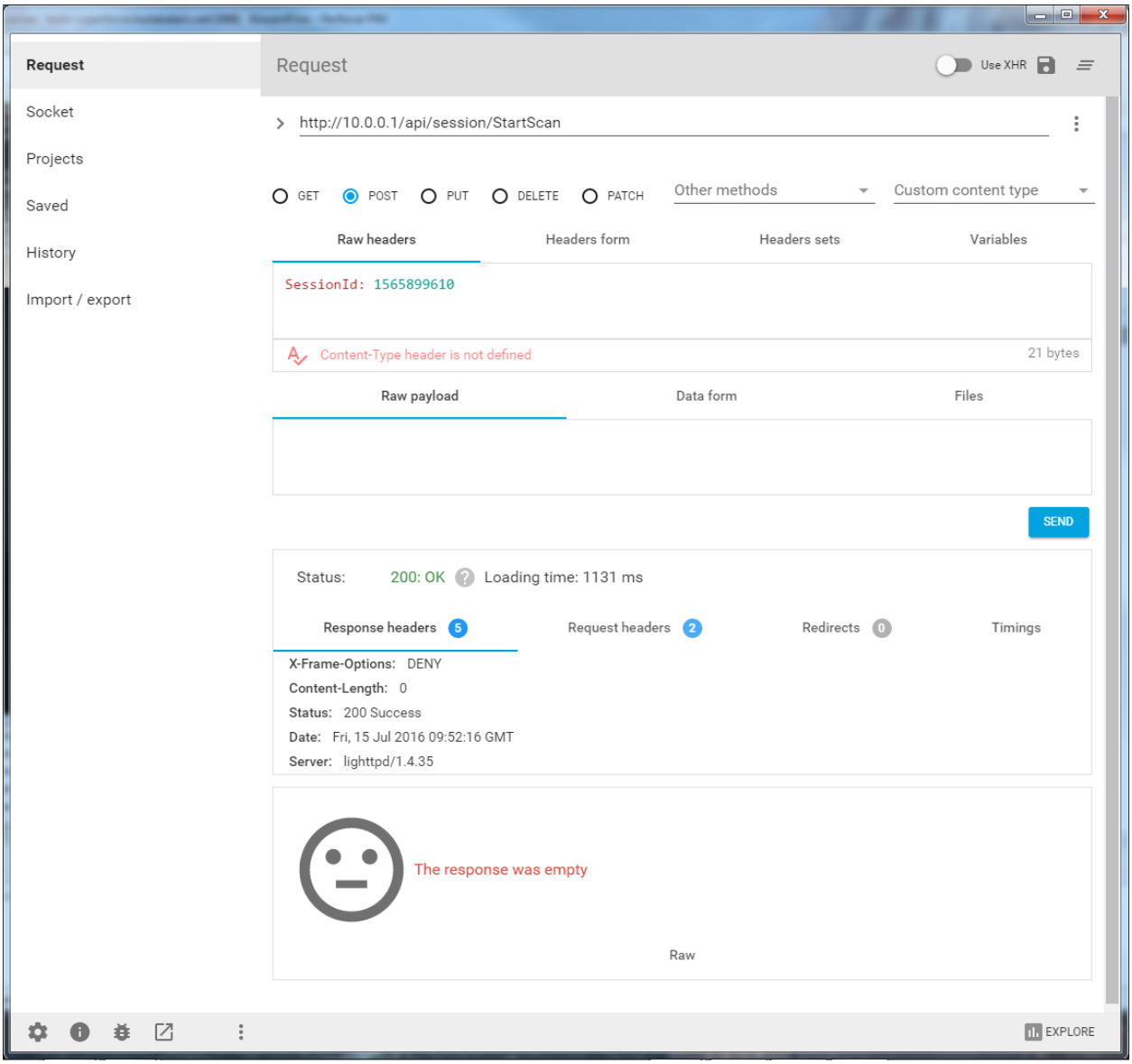
HTTP	API	Returns	Description

PUT	http://10.0.0.1/api/session
	Include JSON data:
	{"Configuration":{
	"DPI":300,
	"ScanSide":"Duplex",
	"ColorMode":"Color",
	"SkipBlankPages":0}
	}
		Nothing	Change the default configuration for this scan.



Start Scan

HTTP	API	Returns	Description
POST	http://10.0.0.1/api/session/StartScan	Nothing	Initiate a scan



Status: 200: OK – Success. The display on the scanner updates.

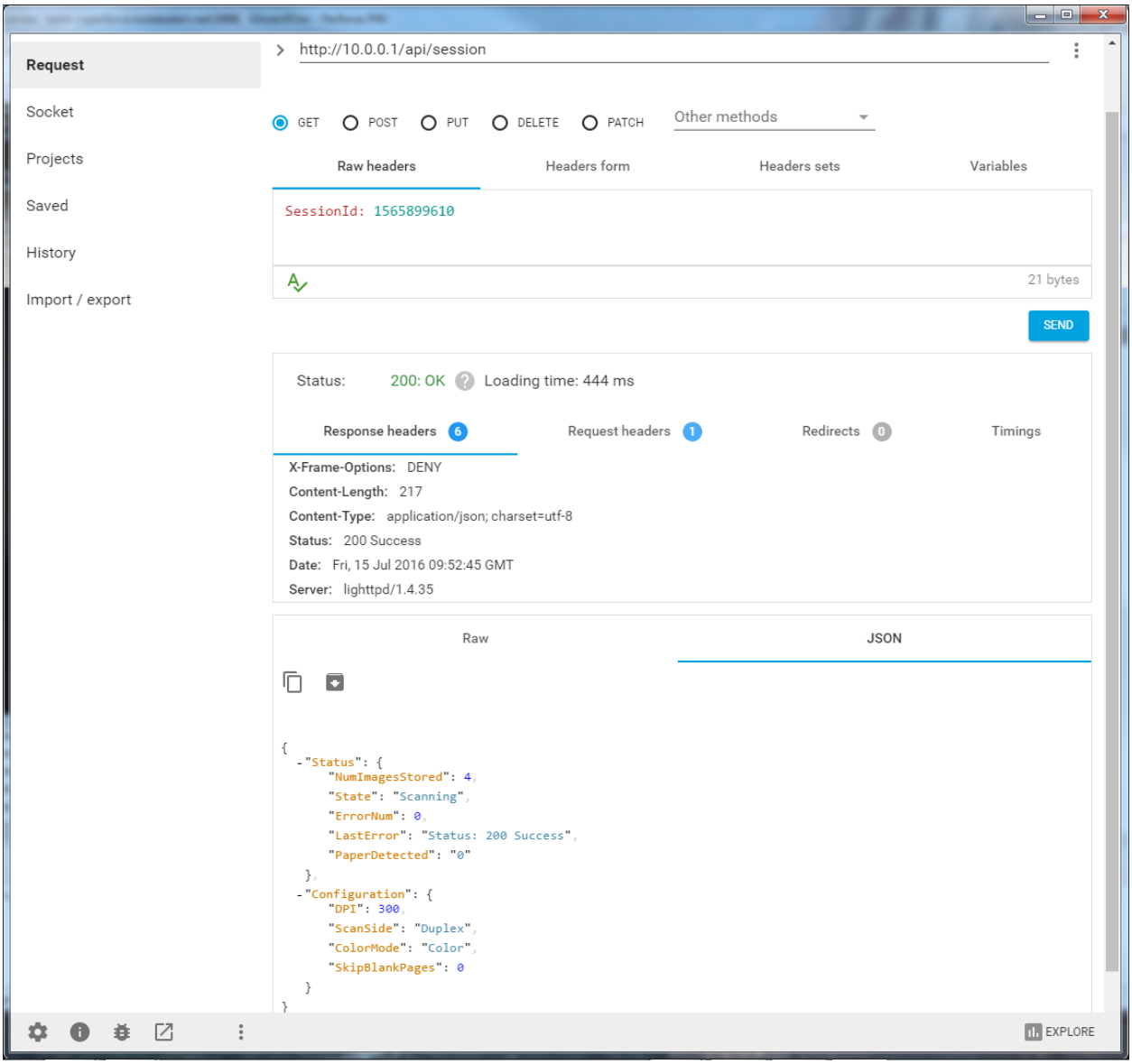
Now put two pages in the scanner and press scan. With no further API calls the scanner will fill its memory with images.

Get Status

HTTP	API	Returns	Description
GET	http://10.0.0.1/api/session	Status and Configuration	Retrieves the scan status and configuration of the scanner and updates the internal state of the session

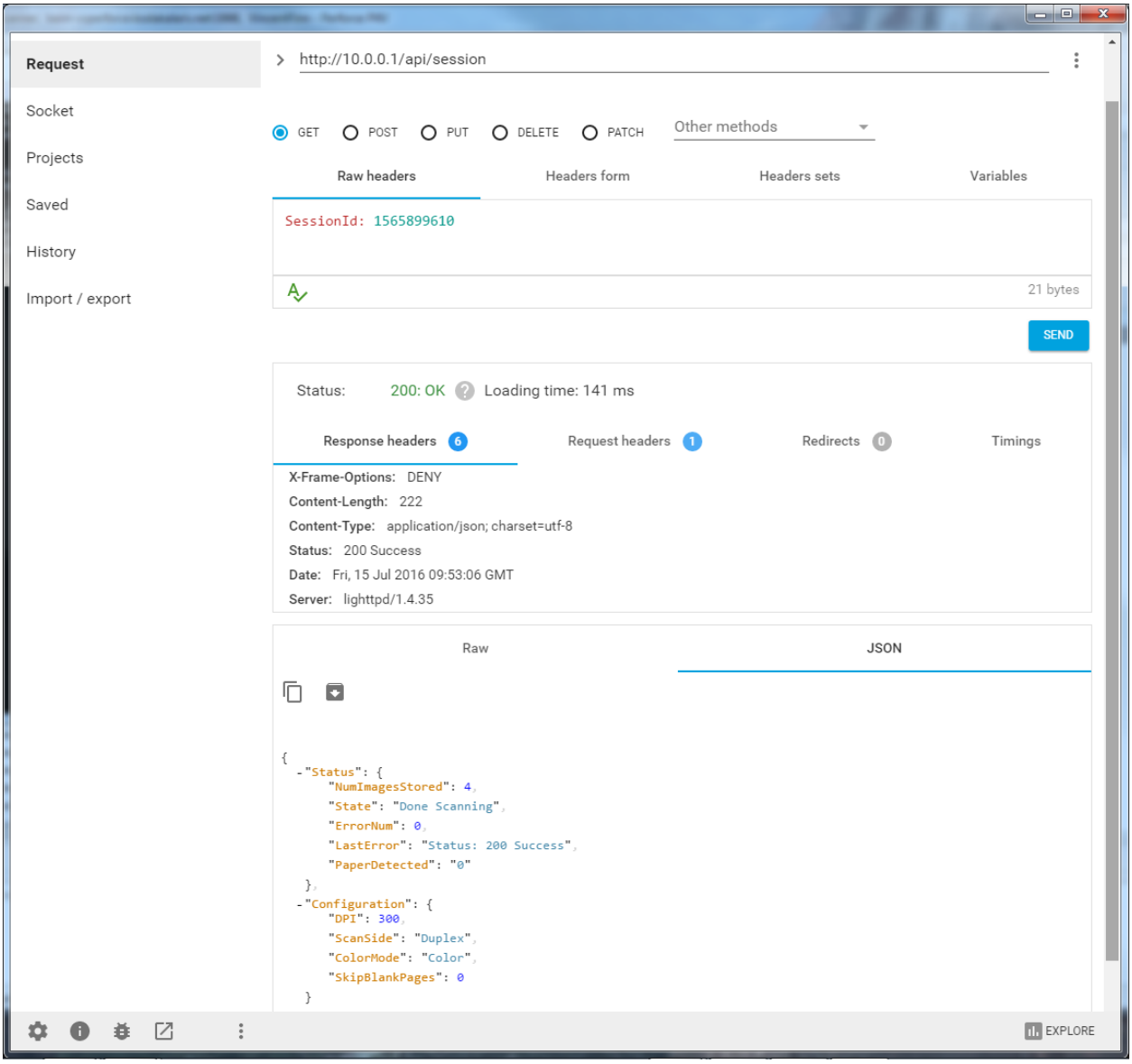
Loop monitoring the Status of the scanner until the State is either “Done Scanning” or “Error”.

Note: If you wait more than five minutes to read the status, the session will time out. In a timeout the OCP clears the message asking you to press Start. If your practice session times out, you’ll need to start over with Create Session.



We see that four pages were scanned, Duplex, Color, 300 DPI and the scanner is still scanning, or waiting for the five second pick timeout.

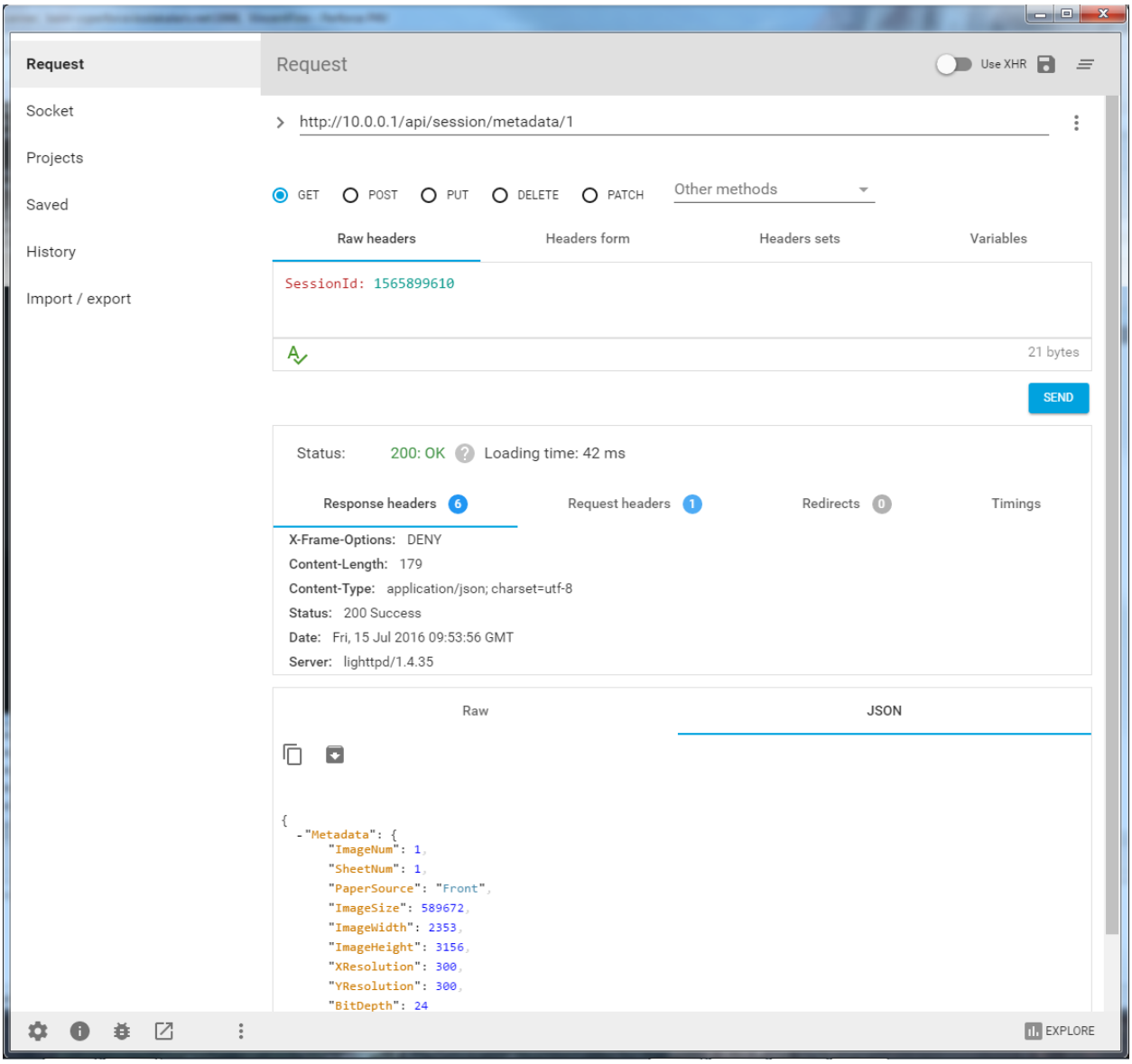
Done Scanning



Eventually the state is “Done Scanning” with a total of 4 images. We did not have to wait for “Done Scanning” before reading the images. As soon as NumImagesStored is incremented, we can read an image or metadata for that image.

Metadata for Scanned Image 1

HTTP	API	Returns	Description
GET	http://10.0.0.1/api/session/metadata/1	JSON data	In JSON format this command retrieves information about image #1.

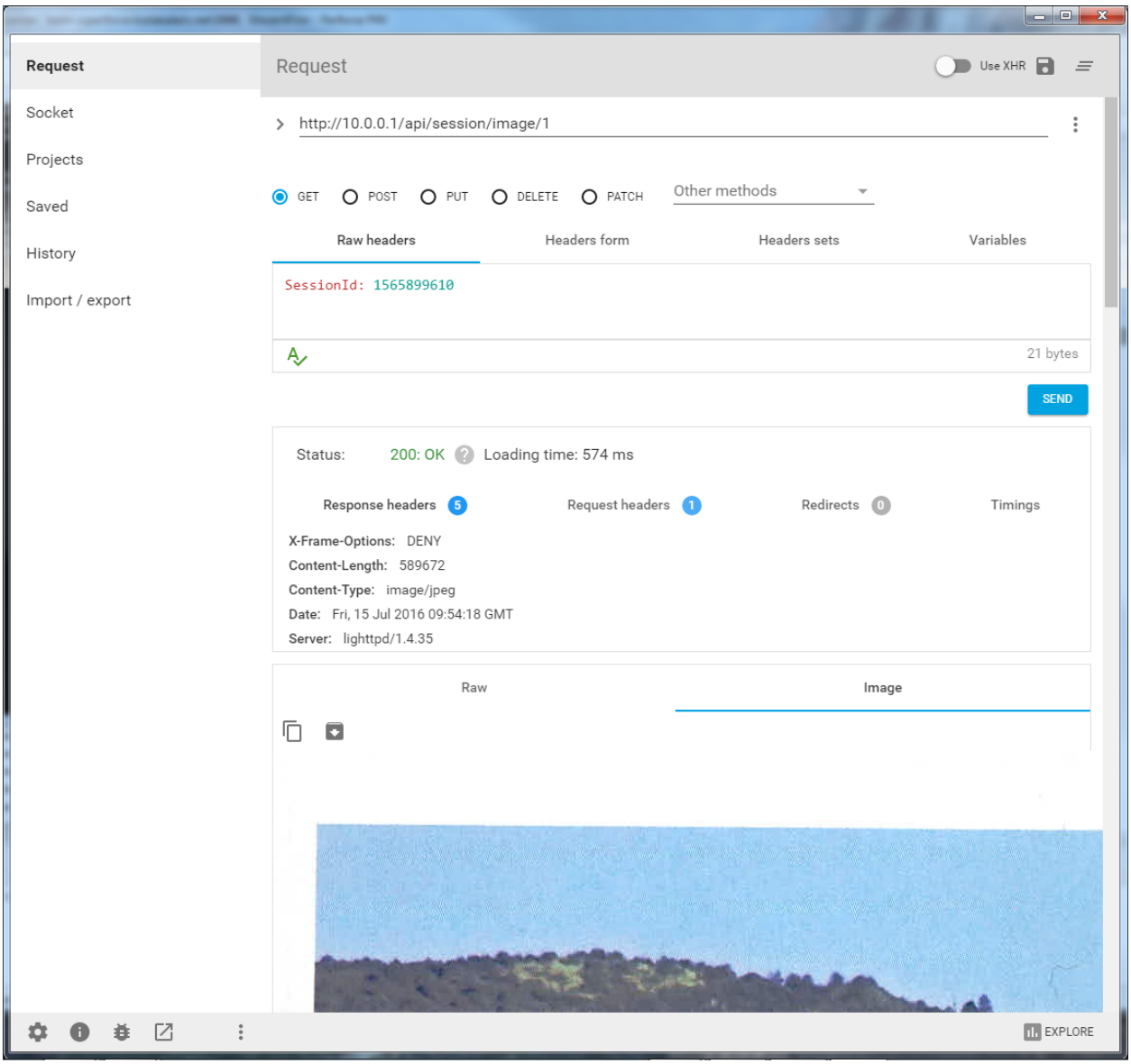


Metadata is optional. Since all image dimensions are also in the image you read, the real value of metadata is when you have SkipBlanks set to “1”. The metadata information returned tells you which SheetNum and whether that image is Front or Back side, to enable you to construct the document.

Read Image 1

While waiting for Done Scanning, you also monitor the NumImagesStored. When it increases you perform the Get Image command shown below.

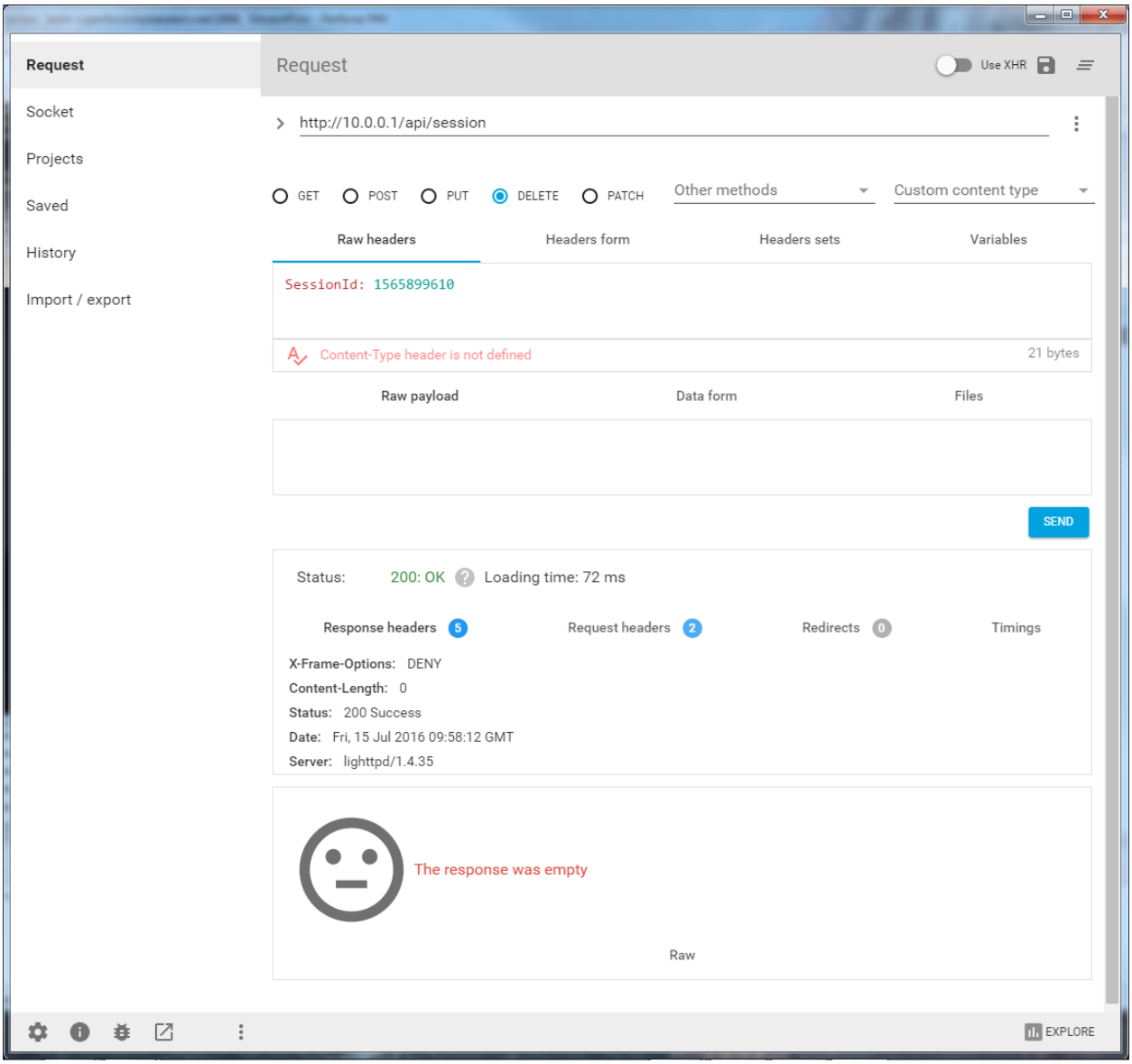
HTTP	API	Returns	Description
GET	http://10.0.0.1/api/session/image/1	JPG or TIF depending on the type of scan.	Read scanned image #1 from the scanner.



Now repeat reading of all images for each page. Notice in the URL that /api/session/image/1 would change to /api/session/image/2 to access the second of the four images.

End Session

HTTP	API	Returns	Description
DELETE	http://10.0.0.1/api/session	Nothing	DONE, all images on the scanner are deleted.



Commands not used in the exercise

Scanner capabilities

In response to the scanner capabilities command, the scanner provides a list of the capabilities that the web API supports. The scanner response also contains the syntax of the commands as you “PUT” the new session configuration to the scanner for the current scan job.

http://10.0.0.1/api/scanner/capabilities?V2

returns:


```
{"Configuration":{
      "DPI":[200,300],
      "ScanSide":["Simplex","Duplex"],
      "SkipBlankPages":[0,1],
      "ColorMode":["Color","Gray","BW"],
      "State":["Idle","In Session","Scanning","Done Scanning","Error"]
}}
```


The above returned information shows all capabilities of the scanner and the settings that can be used to change the scan session parameters. The “scanner” controller exposes changes to the API and shows the ranges, the “schema,” and valid settings.

StopScan

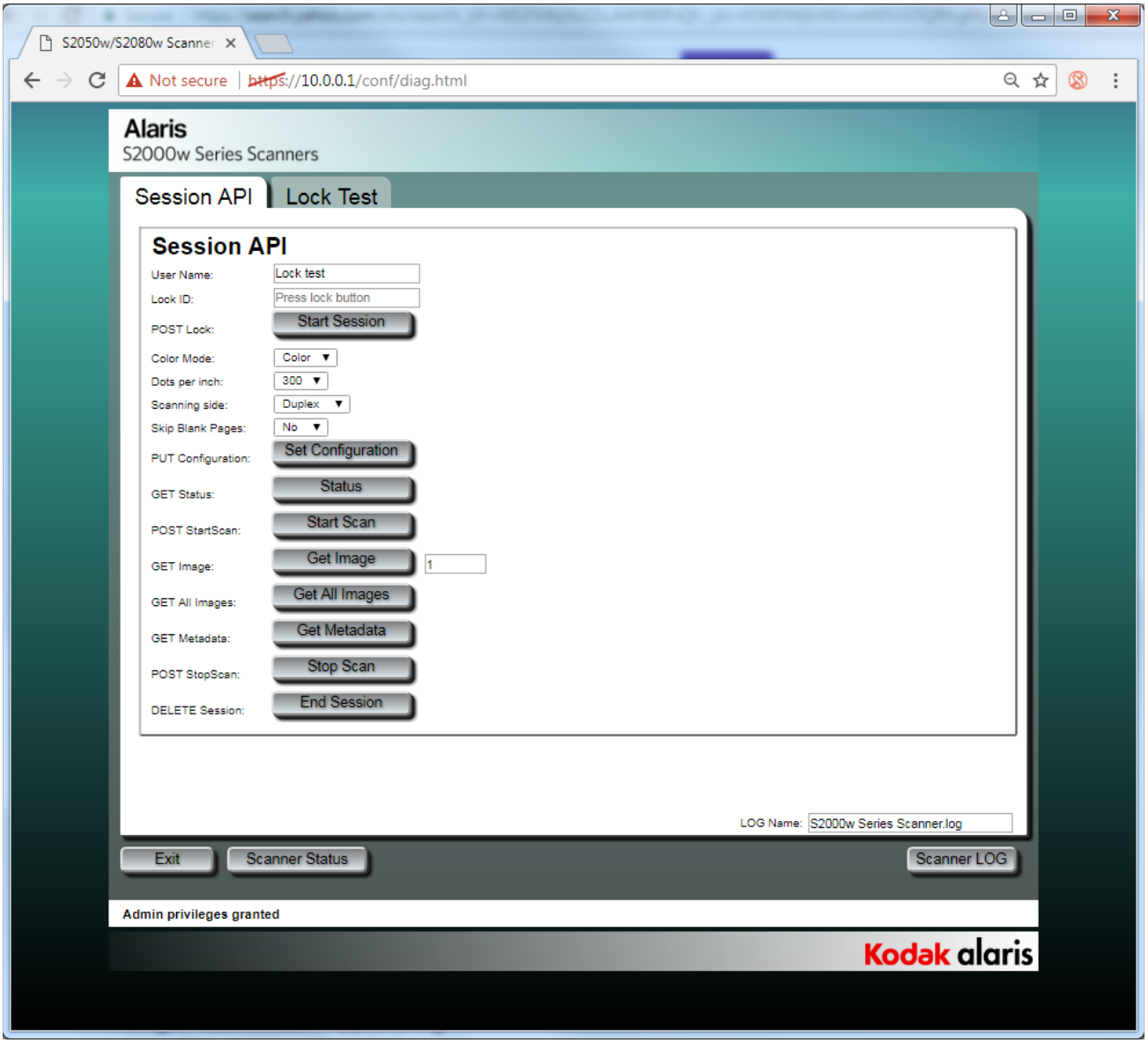
StopScan is a POST call. It stops an actively scanning job at the end of the page that is being scanned. If the job has completed it does nothing. If the job has not completed, it completes the current page.

https://10.0.0.1/api/session/stopscan

After a job completes the state of the scanner is “Done Scanning”. Issuing StopScan during a scan puts the Status->State to “Done Scanning”. In this state you can complete the job by sending and End Session or you can start another job, retaining ownership of the scanner.

Diagnostic Application

The Web API within the scanner can be exercised one API call at a time via a diagnostic webpage on the scanner: http://10.0.0.1/conf/diag.html



The first tab shows each of the API calls with a short description to its left, which informs you whether the button does a GET, POST, DELETE, or PUT. Press F12 to look at the code behind each of these buttons. The web pages and JavaScript within those web pages contain a wealth of information to drive the scanner using the API. Press F12 and explore the code behind this page. Interact with your scanner using this code. Keep in mind that the “Status” button is required to transfer images within the scanner to make them available for download via “Get Image” command.