

CLIENT IMPLEMENTATION GUIDE

WideSCAN S2N API

Acquisition, image retrieval, illumination, and session control for WideSCAN, WideSCAN TUBE SF and LF devices.

Device classes

20 · 22

Device types

115 / 120 · 116 / 121

Interface

HTTP / HTTPS GET

Contents

1. Introduction
2. HTTP and CGI conventions
3. Session control
4. Device discovery
5. Settings
6. Acquisition model
7. The scan command
8. WideSCAN TUBE multi-device acquisition
9. Image and preview retrieval
10. Complete workflows
11. External synchronization
12. Illumination modes
13. Best practices
14. Error handling
15. Quick reference

1. Introduction

The S2N API exposes device control and image acquisition through ordinary HTTP or HTTPS `GET` requests. A client opens an exclusive session, configures the device, starts an acquisition, retrieves image data, and closes the session.

This guide describes the interface from a client implementer's perspective. It covers WideSCAN and WideSCAN TUBE devices in classes 20 and 22.

Supported device families

Device type	Product	Sensor family	Device class
115	WideSCAN	CIS	20
120	WideSCAN TUBE SF	CIS	20
116	WideSCAN LF	CCD	22
121	WideSCAN TUBE LF	CCD	22

Device architecture and intended use

WideSCAN and WideSCAN TUBE devices use the same S2N concepts for discovery, session control, configuration, acquisition, and image retrieval. Their processing architecture and intended applications are different:

Characteristic	WideSCAN and WideSCAN LF	WideSCAN TUBE SF and LF
Architecture	A common processing engine is physically separate from the image-acquiring scan bar.	Each scan-bar segment contains its own processing and network interface. No external controller is required.
Client network interface	One S2N device, available with a 1 Gbit/s or 10 Gbit/s Ethernet interface.	One Ethernet S2N device per 24 inches on SF models and per 12 inches on LF models.
Image output	The common processing engine has access to all sensor images and can stitch the individual sensor segments internally.	The client receives separate, overlapping images from the individual S2N devices. These images are intended to be processed independently.
Primary application	Scanning large areas and delivering a combined image.	Error detection and quality control across wide production processes.
Width and speed	Wider scan bars require more processing power from the common engine and therefore provide a lower maximum acquisition speed than smaller scan bars.	Processing capacity is distributed across the segment devices; the client coordinates their acquisition and retrieves a synchronized set of segment images.

For a WideSCAN device, the client normally communicates with one S2N endpoint and receives an internally combined image. For a WideSCAN TUBE, the client opens and coordinates one S2N session per segment. Chapter 8 describes the required start order and frame-group handling for these multi-device systems.

Functional groups

Discovery

s2ninfo identifies a device and reports capabilities without requiring a session.

Session control

s2nopen creates an exclusive session; **s2nclose** releases it.

Configuration

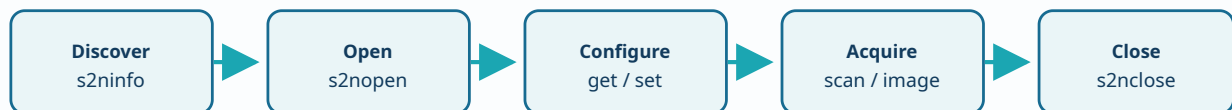
get reads settings and **set** changes them.

Acquisition

scan triggers direct acquisition or controls multiframe operation.

Image data

image returns full-resolution data; **preview** returns a preview representation.



The session ID returned by s2nopen is included in all session-bound requests.

Figure 1 — Typical lifecycle of an S2N client.

2. HTTP and CGI conventions

Functions are invoked as CGI programs below the device's `/cgi/` path. The examples throughout this guide use `192.168.1.50`: this is the default IP address of a WideSCAN device and, for a WideSCAN TUBE, represents the master device. Unless an example explicitly names another TUBE segment, establish the connection to this address on TCP port 80 and send the path as the HTTP request target. The `Host` header is required by HTTP/1.1.

```
GET /cgi/<function>?<argument-1>+<argument-2>+... HTTP/1.1
Host: 192.168.1.50
```

The examples use the normal origin-form request target (`/cgi/...`). A complete URI such as `http://192.168.1.50/cgi/...` is not placed in the request line for a direct connection; that absolute form is normally used when communicating through an HTTP proxy. When HTTPS is enabled, establish the corresponding TLS connection, normally on port 443, while retaining the same request-target format.

Arguments follow the question mark and are separated by a literal plus sign (`+`).

This syntax comes from the CGI command-line convention. It does *not* use ordinary `name=value&name=value` query parameters. Named arguments use a colon:

```
GET /cgi/set?SID+dpi:300+colormode:truecolor HTTP/1.1
Host: 192.168.1.50
```

Do not replace + with &. The plus sign separates CGI arguments. If a literal plus is part of an argument value, percent-encode it as `%2B`.

Text response framing

Text responses use CRLF line endings, the same byte sequence used by HTTP headers. A blank CRLF line terminates the response payload:

```
key:value\r\n
key:value\r\n
\r\n
```

For a textual command response, continue reading CRLF-terminated lines until the empty line is received. The empty line is the S2N response terminator; do not wait for the device to close the HTTP or TCP connection. Clients should parse response bytes deliberately and must not assume platform-native line endings.

Image-returning functions send binary data; always inspect the HTTP `Content-Type` before decoding a response as an image because an error response can be textual.

S2N status responses

The HTTP transport status alone does not indicate whether an S2N operation succeeded. For a textual response, parse the final `Status` field. A successful session-open response also contains the opaque session ID:

```
ID: 12345\r\n
Status: OK\r\n
\r\n
```

Errors and warnings use the same structured format: a severity, a numeric code, and diagnostic text.

Error example

```
Status: ERROR 02: Invalid Session ID\r\n
\r\n
```

Warning example

```
Status: WARNING 160: No white balance data\r\n
\r\n
```

Use the severity and numeric code for program logic. Preserve the diagnostic text for logs, but do not depend on its exact wording.

3. Session control

A session grants exclusive access to the device. Open the session before configuration or acquisition and close it in normal and error paths.

Open a session

```
GET /cgi/s2nopen HTTP/1.1
Host: 192.168.1.50
```

The response contains a session identifier. Treat it as an opaque value and pass it unchanged as the first argument of every session-bound request. Request examples use `SID` as the placeholder for this returned value.

Close a session

```
GET /cgi/s2nclose?SID HTTP/1.1
Host: 192.168.1.50
```

Implementation rule: place session cleanup in a language-level `finally`, deferred cleanup, or equivalent mechanism. An abandoned session can prevent another client from opening the device until it is released or times out.

4. Device discovery with `s2ninfo`

`s2ninfo` is the only function described here that does not need a session ID. Use it before opening a session to identify the device and determine its available capabilities.

```
GET /cgi/s2ninfo HTTP/1.1
Host: 192.168.1.50
```

The function supports information selectors for querying subsets of the device information. A robust client should use reported type, class, dimensions, formats, and setting ranges instead of assuming that all class 20 and class 22 models are identical.

5. Reading and changing settings

Use `get` to inspect the active configuration and supported values, and `set` to update one or more settings. Multiple setting changes may be included in a single request.

```
GET /cgi/get?SID+dpi+colormode+fileformat HTTP/1.1
Host: 192.168.1.50

GET /cgi/set?SID+dpi:300+colormode:truecolor+fileformat:tiff HTTP/1.1
Host: 192.168.1.50
```

A representative response to the `get` request is:

```
dpi: 300 [range] 150 1200\r\n
colormode: truecolor [enum] truecolor indexed8 grayscale lineart photo\r\n
fileformat: tiff [enum] jpeg pnm tiff tf8 pdf png bmp\r\n
Status: OK\r\n
\r\n
```

The current value follows the setting name. The `[range]` or `[enum]` marker and the remaining fields describe the values reported by that device. Clients should parse the returned capabilities instead of assuming that this example applies to every firmware version.

Supported values and ranges are device- and firmware-dependent. Query them rather than embedding limits in the client. Configure persistent acquisition settings before starting multiframe mode.

Defining the physical frame area

Set `docsize:user` to define a custom physical area for every acquired frame. The `user_x` and `user_y` settings specify the area's origin; `user_w` and `user_h` specify its width and length.

```
GET /cgi/set?SID+docsize:user+user_x:0+user_y:0+user_w:8000+user_h:12000 HTTP/1.1
Host: 192.168.1.50
```

All four values are expressed in units of 1/1000 inch. In this example, the physical frame area is 8 inches wide and 12 inches long. These settings determine the physical dimensions acquired for each frame, including each frame generated in multiframe mode.

Frame area is not an ROI. The `user_x`, `user_y`, `user_w`, and `user_h` settings define the physical acquisition area. The `image_x`, `image_y`, `image_w`, and `image_h` arguments select a region from an already acquired frame during image or preview retrieval.

The maximum length of an individual frame segment is device-dependent. WideSCAN devices support longer frames than WideSCAN TUBE devices; consult the datasheet for the applicable device limit. The maximum is determined by the device's internal image-memory capacity. For best performance in multiframe mode, keep the frame length at or below 12 inches (approximately 305 mm).

Virtual-rescan settings

Some settings change how stored sensor data is rendered rather than causing a new physical exposure. These virtual-rescan settings cannot be changed while the scan mode is `multiframe`.

1. Exit multiframe mode.
2. Apply the required setting changes.
3. Restart multiframe acquisition.

ROI requests are different. Request-local region arguments used by image or preview remain useful for retrieving several regions from the same queued frame. They do not require advancing the queue.

6. Acquisition model

S2N on WideSCAN devices offers two acquisition styles:

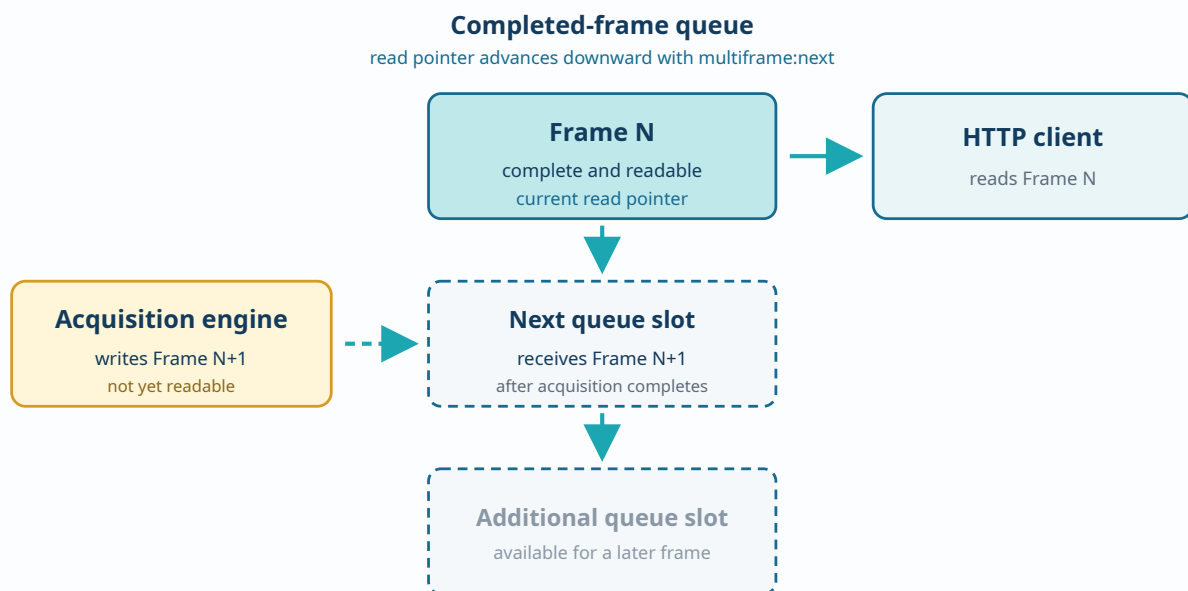
Direct acquisition

A plain `scan` request performs one acquisition. Use it for an isolated scan where continuous throughput is unnecessary.

Multiframe acquisition

`multiframe:start` starts acquisition internally. Completed frames enter an internal queue and can be retrieved independently of the acquisition timing.

Multiframe mode decouples acquisition from HTTP image transfer and is the preferred mode for repeated scans. A completed frame becomes available to the client immediately; acquisition of the following frame may continue in parallel. A frame that is still being written is not readable from the queue.



The client can read Frame N while the acquisition engine writes Frame N+1.

Figure 2 — The client reads only completed queued frames. While Frame N is being transferred, the device may acquire Frame N+1 in a separate write operation. The first frame can be fetched as soon as it is complete.

The queue is finite. Retrieve and advance frames promptly. A client should not start multiframe mode and then leave the queue unattended.

7. The scan command

Single scan

```
GET /cgi/scan?SID HTTP/1.1  
Host: 192.168.1.50
```

This starts one direct acquisition. Use it when the application needs exactly one image and does not benefit from a continuously filled queue.

Start multiframe acquisition

```
GET /cgi/scan?SID+multiframe:start HTTP/1.1  
Host: 192.168.1.50
```

This switches to multiframe mode, starts acquisition internally, and queues completed images. The client retrieves queued images with `image` or `preview`.

Stop multiframe acquisition

```
GET /cgi/scan?SID+multiframe:stop HTTP/1.1  
Host: 192.168.1.50
```

This stops the active acquisition while retaining multiframe scan mode. It is useful when acquisition is to be paused without changing the operating mode.

Exit multiframe mode

```
GET /cgi/scan?SID+multiframe:exit HTTP/1.1  
Host: 192.168.1.50
```

This stops multiframe operation and returns the device to direct scan mode. Use it before changing virtual-rescan settings or switching to single-scan behavior.

8. WideSCAN TUBE multi-device acquisition

WideSCAN TUBE systems are arrays of internal WideSCAN devices. Each internal device is an independent network endpoint with its own S2N session, settings, scan command, and image data. The client must coordinate these endpoints so that every segment is ready when synchronized acquisition begins.

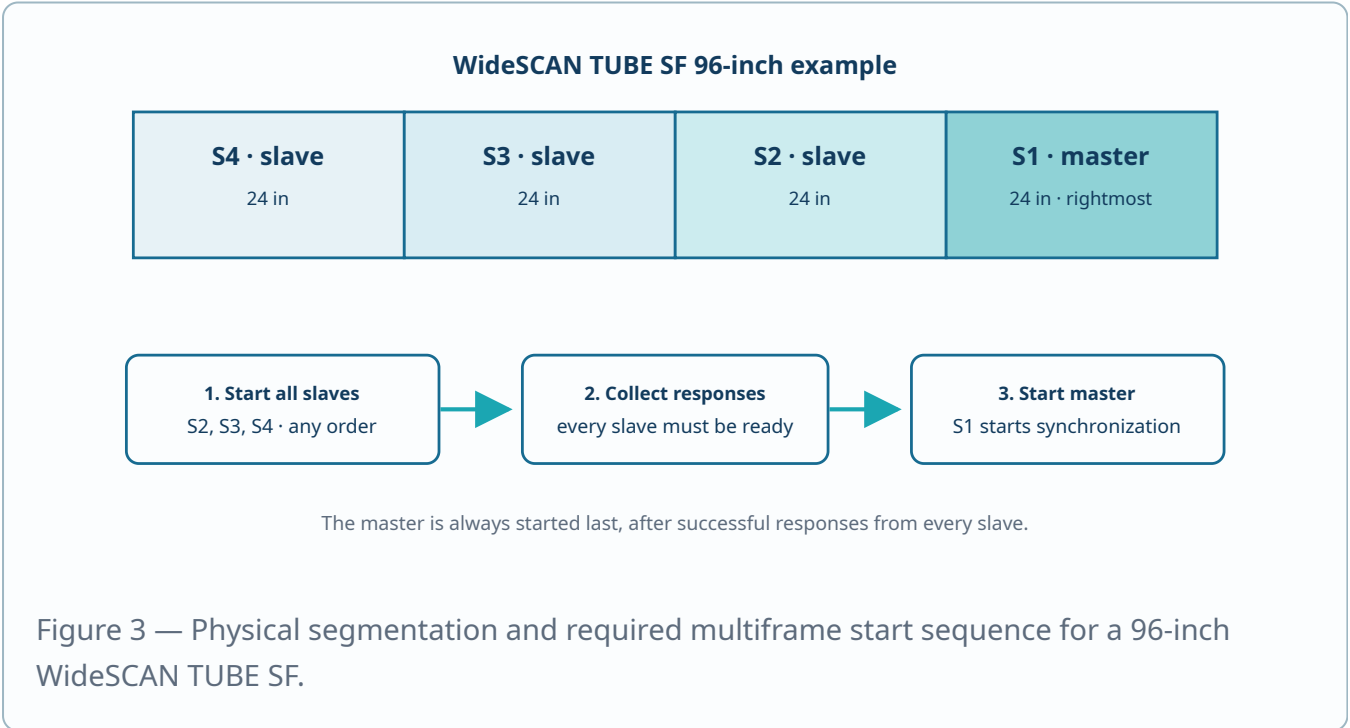
Segment layout and naming

A WideSCAN TUBE SF contains one network device for every 24 inches of scan width. Segment host names use the suffixes `S1` through `S8`, depending on the configured system width. `WST-SF-S1` is the master and is the rightmost device. The remaining endpoints are slaves.

Across the complete WideSCAN TUBE, segment order is left to right as `SN → ... → S2 → S1`. In other words, segment numbers decrease toward the right, and `S1` is always the rightmost segment.

A 96-inch WideSCAN TUBE SF therefore contains four internal devices:

Host name	Role	Physical segment
<code>WST-SF-S1</code>	Master	Rightmost 24 inches
<code>WST-SF-S2</code>	Slave	24 inches
<code>WST-SF-S3</code>	Slave	24 inches
<code>WST-SF-S4</code>	Slave	Leftmost 24 inches



Recommended multiframe start sequence

Multiframe mode is the recommended acquisition mode for WideSCAN TUBE devices. Configure and open a separate session on every internal device, then start acquisition as follows:

1. Send `scan` with `multiframe:start` to all slave devices (`S2` through `SN` ; up to `S8`). Slave order is not important; the requests may be issued concurrently.
2. Collect and validate the `scan` response from every slave.
3. Only after every slave has reported success, send `scan` with `multiframe:start` to the master, `S1` .

This ordering ensures that all slaves are ready to acquire scan lines when the master generates—or receives—the first line-sync pulse.

```
# Step 1: start the three slaves in any order or concurrently
GET /cgi/scan?SESSION_S2+multiframe:start HTTP/1.1
Host: WST-SF-S2

GET /cgi/scan?SESSION_S3+multiframe:start HTTP/1.1
Host: WST-SF-S3

GET /cgi/scan?SESSION_S4+multiframe:start HTTP/1.1
Host: WST-SF-S4

# Step 2: wait for and validate all three slave responses

# Step 3: start the master last
GET /cgi/scan?SESSION_S1+multiframe:start HTTP/1.1
Host: WST-SF-S1
```

Do not start the master after only dispatching the slave requests. The client must receive successful responses from all slaves first. If any slave fails to become ready, do not start the master; cleanly stop the devices already started before retrying.

Retrieving synchronized frame groups

One synchronized TUBE acquisition produces a logical frame group containing the current queued frame from every internal device. For the 96-inch SF example:

```
Logical frame N
├─ WST-SF-S4 – leftmost segment
├─ WST-SF-S3
├─ WST-SF-S2
└─ WST-SF-S1 – rightmost segment / master
```

Treat this group as one indivisible application frame. Segment transfers may run concurrently, but each device has its own queue and read pointer. Within each segment, all requests for the current frame must be

performed in their required order.

1. For every segment, retrieve the complete frame or all required ROIs and preview representations.
2. Requests that are not the last transfer for that segment and frame must omit `multiframe:next`.
3. The very last `image` or `preview` transfer for that segment and frame must include `multiframe:next`. The response still contains the current frame; the segment queue advances after that response.
4. Advance every segment exactly once for each logical frame group.
5. Publish or process the logical TUBE frame only after the required transfers from all segments have completed successfully.

Segment	Earlier transfers from frame N	Last transfer from frame N
S4	<code>image</code> / <code>preview</code> without <code>multiframe:next</code>	<code>image</code> / <code>preview</code> with <code>multiframe:next</code>
S3	<code>image</code> / <code>preview</code> without <code>multiframe:next</code>	<code>image</code> / <code>preview</code> with <code>multiframe:next</code>
S2	<code>image</code> / <code>preview</code> without <code>multiframe:next</code>	<code>image</code> / <code>preview</code> with <code>multiframe:next</code>
S1	<code>image</code> / <code>preview</code> without <code>multiframe:next</code>	<code>image</code> / <code>preview</code> with <code>multiframe:next</code>

Frame-group summary: retrieve everything required from frame N on each segment, put `multiframe:next` on that segment's final transfer, wait for all segments, and only then handle frame N as a complete TUBE image.

Keep all segment queues aligned. If an interrupted request or segment failure leaves uncertainty about which queues advanced, discard the incomplete logical frame, stop and exit `multiframe` mode on every internal device, and restart the complete TUBE acquisition sequence.

Segment geometry and image overlap

WideSCAN TUBE output consists of overlapping segment images. The physical pitch between adjacent segment positions is 12 inches, corresponding to 7200 pixels at 600 dpi. Each segment image is wider than this pitch and includes additional pixels at both horizontal edges:

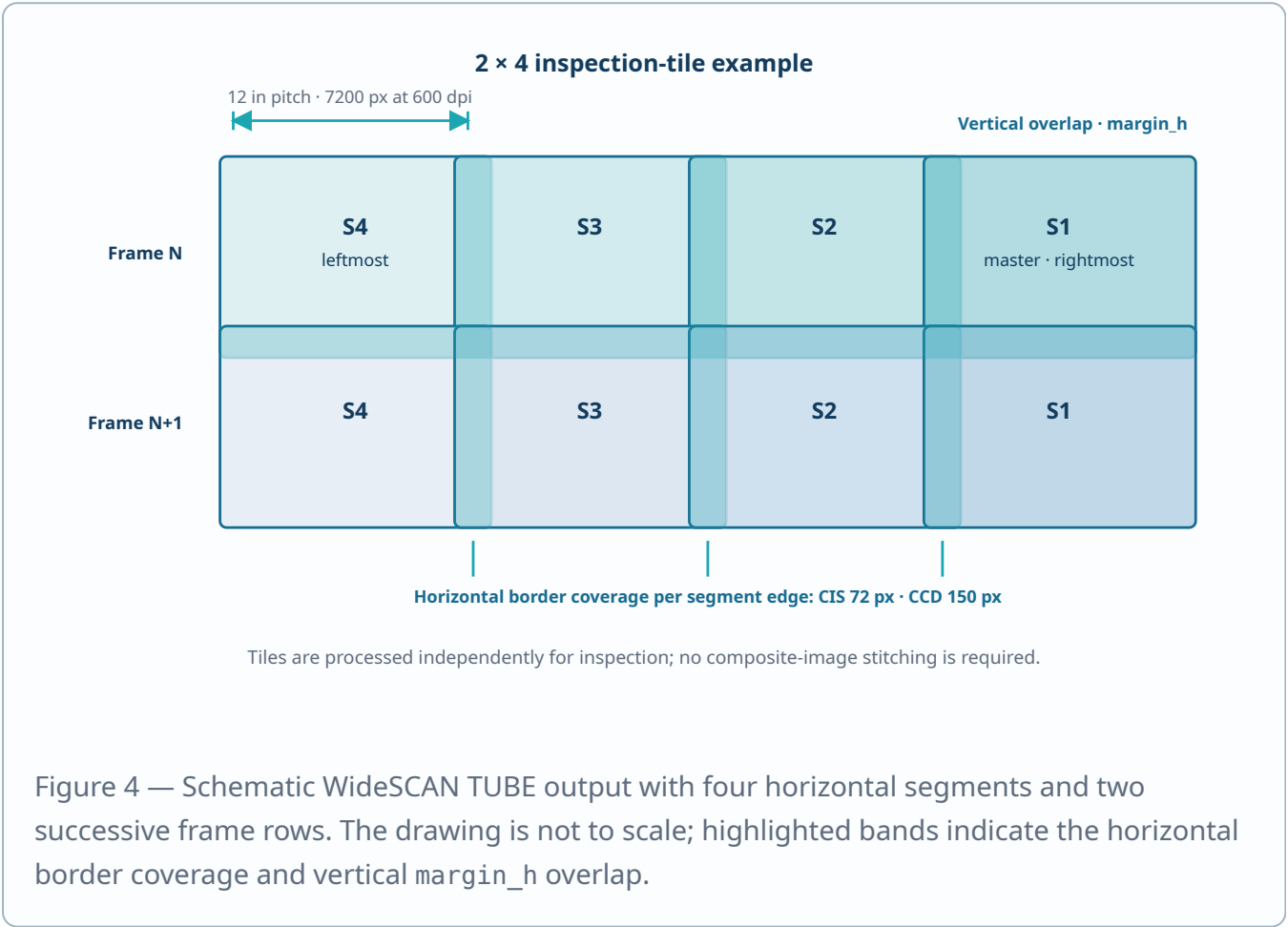
Sensor technology	Segment width at 600 dpi	Width in relation to 12-inch pitch	Additional coverage at each edge
CIS	7344 pixels	$7200 + 2 \times 72$	72 pixels
CCD	7500 pixels	$7200 + 2 \times 150$	150 pixels

At each segment boundary, both neighboring images contain border coverage of the same physical area. Each participating CIS segment contributes 72 pixels beyond its nominal edge; each participating CCD segment contributes 150 pixels.

Vertical overlap is controlled by `margin_h`, expressed in units of 1/1000 inch. The setting adds an overlap margin to the physical frame height. For example:

```
GET /cgi/set?SID+docsize:user+user_h:10200+margin_h:200 HTTP/1.1
Host: 192.168.1.50
```

This configuration acquires 10.2 inches in the vertical direction for each tile and advances by 10 inches between successive frame rows, producing a 0.2-inch overlap.



Inspection images, not stitching input: WideSCAN TUBE segment images are intended for error detection and quality-control processing as overlapping tiles. Clients do not need to crop the overlaps, perform geometric correction, or stitch the tiles into one composite image.

Direct single-scan mode WideSCAN TUBE SF

Direct single scans are not recommended on WideSCAN TUBE devices. If direct mode must be used, requests must reach the internal devices in strict reverse segment order, with the master last. For the 96-inch SF example, the required order is:

```
WST-SF-S4 → WST-SF-S3 → WST-SF-S2 → WST-SF-S1 (master)
```

More generally, the order is `SN → ... → S2 → S1`; an eight-segment system therefore starts with `S8` and ends with `S1`.

In direct mode, the `scan` request returns only after that device's scan has completed. The client therefore cannot wait for one response before issuing the next request. It must use asynchronous I/O or separate threads to initiate the blocking requests in the required order while allowing them to remain active concurrently. The implementation must preserve dispatch order despite this concurrency.

Direct single-scan mode WideSCAN TUBE LF

The same master/slave ordering applies to WideSCAN TUBE LF devices, but each internal device covers 12 inches instead of 24 inches. A four-device arrangement therefore corresponds to a 48-inch WideSCAN TUBE LF:

```
WST-LF-S4 → WST-LF-S3 → WST-LF-S2 → WST-LF-S1 (master)
```

For multiframe operation, start the LF slaves in any order, collect every slave response, and then start `WST-LF-S1`. For a direct single scan, initiate requests in strict descending segment order and start `S1` last.

9. image and preview

`image` returns image data at the requested production settings. `preview` returns a preview representation. Both functions can address the current queued frame and accept request-local image-region arguments.

Retrieve the complete frame — standard case

```
GET /cgi/image?SID HTTP/1.1
Host: 192.168.1.50

GET /cgi/preview?SID HTTP/1.1
Host: 192.168.1.50
```

Without ROI arguments, `image` or `preview` returns the complete current frame. Without `multiframe:next`, the queue's read pointer remains on that frame.

File format and request-local overrides

`preview` uses the configured `fileformat` in the same way as `image`. A request may override the format locally without changing the persistent setting. For example, a client can request a JPEG preview even when another format is configured:

```
GET /cgi/preview?SID+fileformat:jpeg HTTP/1.1
Host: 192.168.1.50
```

The override applies only to this response. The same request-local mechanism can be used with `image`.

The following three examples apply to `scanmode:multiframe`. They operate on the current frame in the internal image queue. Queue advancement with `multiframe:next` is not part of direct single-scan operation.

Multiframe: retrieve the complete frame and advance

```
GET /cgi/image?SID+multiframe:next HTTP/1.1
Host: 192.168.1.50

GET /cgi/preview?SID+multiframe:next HTTP/1.1
Host: 192.168.1.50
```

The complete current frame is returned first; the read pointer is then advanced to the next queued frame.

Multiframe: retrieve an ROI without advancing

```
GET /cgi/image?SID+image_x:0+image_y:0+image_w:1024+image_h:512 HTTP/1.1
Host: 192.168.1.50
```

Without `multiframe:next`, the call reads from the current queued frame and leaves the queue's read pointer unchanged. The client can therefore request several regions of interest, formats, or preview representations derived from the same acquisition.

Multiframe: retrieve an ROI and advance

```
GET /cgi/image?SID+image_x:0+image_y:0+image_w:1024+image_h:512+multiframe:next HTTP/1.1
Host: 192.168.1.50
```

The current frame is returned first; the read pointer is then advanced to the next queued frame. Put `multiframe:next` on the final retrieval associated with a frame.

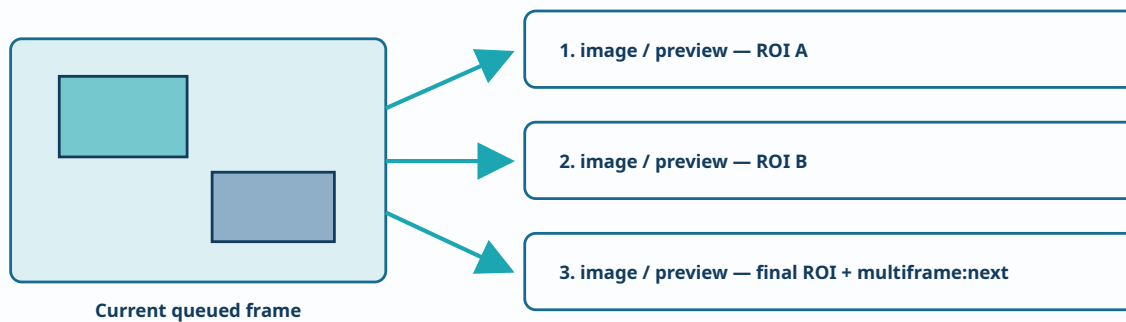


Figure 5 — Several requests can use the same physical acquisition. Advance the queue only on the last retrieval.

Binary response format

A successful `image` or `preview` request returns the encoded file directly after the HTTP header. It does not prefix the binary data with an S2N `Status: OK` line. If the request fails, the response uses `Content-Type: text/plain` and contains an S2N error or warning response as described in Chapter 2.

fileformat	Successful response Content-Type
<code>jpeg</code>	<code>image/jpeg</code>
<code>pnm</code> , grayscale	<code>image/x-portable-graymap</code>
<code>pnm</code> , color	<code>image/x-portable-pixmap</code>
<code>pnm</code> , bitonal	<code>image/x-portable-bitmap</code>
<code>tiff</code> or <code>tf8</code>	<code>image/tiff</code>
<code>png</code>	<code>image/png</code>
<code>bmp</code>	<code>image/bmp</code>
<code>pdf</code>	<code>application/pdf</code>

Determine the actual image width and height from the encoded image header. Do not derive the returned dimensions solely from configured frame or ROI values: internal rounding can change a dimension, and image-processing operations may produce different dimensions.

For `tiff` and `tf8`, determine byte order from the TIFF header. Current WideSCAN firmware writes little-endian TIFF data, but clients should follow the header rather than assuming a fixed byte order.

Use the HTTP message framing provided by the client library to determine the response-body boundary; do not infer the byte count from the requested image dimensions.

10. Complete client workflows

Workflow A — One isolated scan

```
# Discover the device; no session is required
GET /cgi/s2ninfo HTTP/1.1
Host: 192.168.1.50

# Open and retain the returned session ID
GET /cgi/s2nopen HTTP/1.1
Host: 192.168.1.50

# Configure the acquisition
GET /cgi/set?SESSION+dpi:300+colormode:truecolor+illumination:top HTTP/1.1
Host: 192.168.1.50

# Perform one acquisition
GET /cgi/scan?SESSION HTTP/1.1
Host: 192.168.1.50

# Retrieve the result
GET /cgi/image?SESSION HTTP/1.1
Host: 192.168.1.50

# Always release the session
GET /cgi/s2nclose?SESSION HTTP/1.1
Host: 192.168.1.50
```

This is the simplest sequence and is appropriate when an application needs only a single acquisition.

Workflow B — Repeated multiframe acquisition

```
GET /cgi/s2nopen HTTP/1.1
Host: 192.168.1.50

GET /cgi/set?SESSION+dpi:300+colormode:truecolor+illumination:split HTTP/1.1
Host: 192.168.1.50

GET /cgi/scan?SESSION+multiframe:start HTTP/1.1
Host: 192.168.1.50

# Repeat while acquisition is active:
GET /cgi/image?SESSION+image_x:0+image_y:0+image_w:800+image_h:600 HTTP/1.1
Host: 192.168.1.50

GET /cgi/image?SESSION+image_x:800+image_y:0+image_w:800+image_h:600+multiframe:next HTTP/1.1
Host: 192.168.1.50

# Controlled shutdown:
GET /cgi/scan?SESSION+multiframe:stop HTTP/1.1
Host: 192.168.1.50

GET /cgi/scan?SESSION+multiframe:exit HTTP/1.1
Host: 192.168.1.50

GET /cgi/s2nclose?SESSION HTTP/1.1
Host: 192.168.1.50
```

The first image request reads one region without advancing. The second request reads another region from that same frame and then advances to the next frame.

Workflow C — Reprocess a scan with a virtual-rescan setting

```
# Acquire one frame in direct scan mode
GET /cgi/scan?SESSION HTTP/1.1
Host: 192.168.1.50

# Retrieve the image with the current settings
GET /cgi/image?SESSION HTTP/1.1
Host: 192.168.1.50

# Change brightness without acquiring another frame
GET /cgi/set?SESSION+bright:255 HTTP/1.1
Host: 192.168.1.50

# Retrieve the same acquired frame with the new brightness
GET /cgi/image?SESSION HTTP/1.1
Host: 192.168.1.50
```

The second `image` request reprocesses the data from the same physical acquisition using the changed brightness setting. No additional `scan` request is required.

11. External synchronization

WideSCAN CIS devices can synchronize acquisition with external equipment through the encoder input. The synchronization settings control the line clock (LSYNC), the acquisition-enable signal (PSYNC), and optional internal scaling of the external line clock.

Availability by device

Device	Type	Available synchronization settings
WideSCAN	115	<code>lsync_mode</code> , <code>psync_mode</code> , <code>lsync_scaling</code> , <code>free_running_speed</code>
WideSCAN TUBE SF	120	<code>lsync_mode</code> , <code>psync_mode</code> , <code>lsync_scaling</code> , <code>free_running_speed</code>
WideSCAN LF	116	<code>free_running_speed</code> only
WideSCAN TUBE LF	121	<code>free_running_speed</code> only

Settings

Setting	Values	Purpose
<code>lsync_mode</code>	<code>free_running</code> , <code>dual_edge</code>	Selects the internal line clock or external line triggering on both encoder-signal edges.
<code>psync_mode</code>	<code>free_running</code> , <code>active_high</code>	Selects continuous acquisition or acquisition enabled by the PSYNC input.
<code>lsync_scaling</code>	<code>off</code> , <code>auto</code>	Controls automatic internal division of the external LSYNC signal.
<code>free_running_speed</code>	Device-dependent range	Sets the nominal free-running scan speed.

Free-running speed

`free_running_speed` represents the physical scan speed in inches per second. The API value is the speed in in/s multiplied by 1000:

Speed conversion: $\text{free_running_speed} = \text{speed [in/s]} \times 1000$. For example, 64000 selects 64 in/s.

```
GET /cgi/set?SID+free_running_speed:64000 HTTP/1.1
Host: 192.168.1.50
```

Read the active value with `get`:

```
GET /cgi/get?SID+free_running_speed HTTP/1.1
Host: 192.168.1.50
```

A representative response for a configured speed of 64 in/s is:

```
free_running_speed: 64000\r\n
Status: OK\r\n
\r\n
```

Use the device datasheet for speed limits. The usable range is device-dependent and is not currently represented by model-specific S2N range metadata. Do not infer that every value accepted by the API is valid for every WideSCAN device.

Synchronization modes on CIS devices

lsync_mode	psync_mode	Behavior
free_running	free_running	No external hardware synchronization. Scan speed is determined by <code>free_running_speed</code> .
free_running	active_high	The internal line clock is used, but acquisition proceeds only while PSYNC is logic high. Acquisition pauses while PSYNC is logic low.
dual_edge	free_running	Every rising and falling edge at the encoder input triggers a new scan sub-line. PSYNC does not gate acquisition.
dual_edge	active_high	External encoder edges trigger scan sub-lines while PSYNC is logic high. Acquisition pauses while PSYNC is logic low.

With `psync_mode:active_high`, PSYNC acts as an acquisition enable or pause signal; it does not define the requested frame length. The length remains determined by `docsize` and, for `docsize:user`, by the configured `user_h`. Configure the physical frame area before starting acquisition. Holding PSYNC low and then driving it high can therefore be used as an external scan-start mechanism.

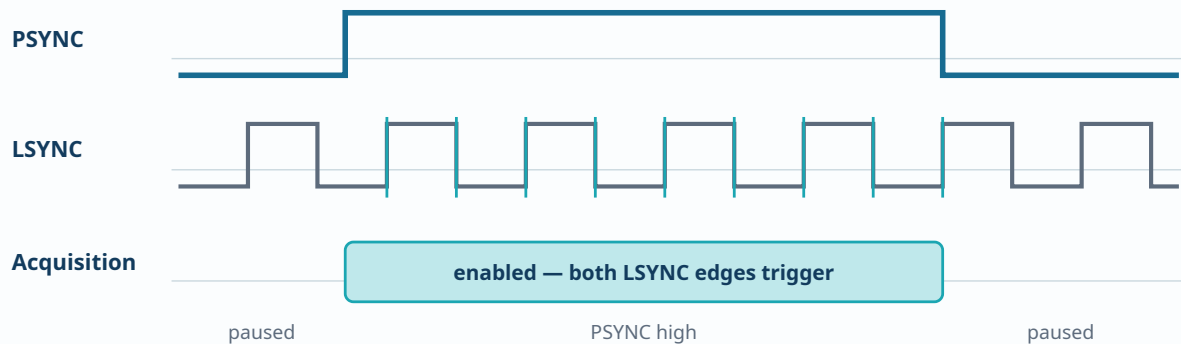


Figure 6 — Combined `psync_mode:active_high` and `lsync_mode:dual_edge`. LSYNC edges are accepted while PSYNC is high; acquisition pauses while PSYNC is low.

```
GET /cgi/set?SID+lsync_mode:free_running+psync_mode:active_high HTTP/1.1
Host: 192.168.1.50
```

PSYNC gating can also be combined with external line synchronization:

```
GET /cgi/set?SID+lsync_mode:dual_edge+psync_mode:active_high+lsync_scaling:auto HTTP/1.1
Host: 192.168.1.50
```

External LSYNC frequency

In `dual_edge` mode, both edges of the encoder signal are used. Define the calculated output-line frequency as:

Calculated line frequency: $f_{\text{line}} = \text{resolution [dpi]} \times \text{transport speed [in/s]}$

When `lsync_scaling:off`, choose the encoder input frequency for the configured color mode:

Color mode	Edges per output line	Required encoder input frequency
RGB	3	$f_{\text{encoder}} = 3/2 \times f_{\text{line}}$
Grayscale	1	$f_{\text{encoder}} = 1/2 \times f_{\text{line}}$

RGB acquisition requires three internal CIS sub-lines for each RGB output line. Because `dual_edge` uses two edges per encoder cycle, the encoder frequency is 3/2 of the calculated output-line frequency. Grayscale needs one edge per output line, resulting in 1/2 of the calculated output-line frequency.

Automatic LSYNC scaling

With `lsync_scaling:auto`, the external encoder can operate at the frequency calculated for 600 dpi RGB, independent of the selected lower resolution or grayscale mode. The device applies the required division internally:

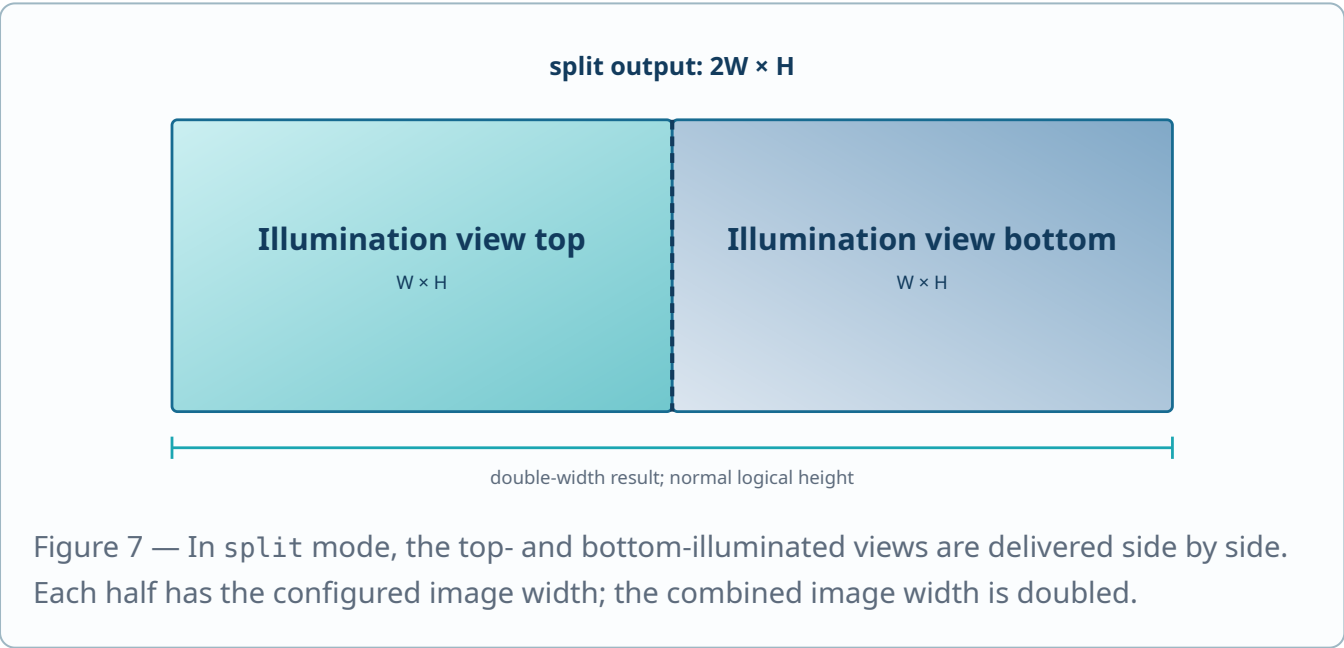
Automatic-scaling input frequency: $f_{\text{encoder}} = 3/2 \times 600 \times \text{transport speed [in/s]} = 900 \times \text{transport speed [Hz]}$

For example, at a transport speed of 40 in/s, the encoder input frequency with automatic scaling is $900 \times 40 = 36,000 \text{ Hz}$. Consult the device datasheet for electrical levels, timing tolerances, and the supported encoder-frequency range.

12. Illumination modes

The device supports complementary illumination from the top and bottom lamp banks. The active mode determines whether the client receives one illuminated view, a calculated result, or both views in one image.

Mode	Output	Typical use
default	Device's normal illumination configuration	General acquisition
top	Image illuminated from the top direction	Directional surface inspection
bottom	Image illuminated from the bottom direction	Complementary directional inspection
3d	A calculated image derived from the sums and differences of the top and bottom views	Emphasizing surface-height and shape information
split	Top and bottom illuminated views placed side by side	Applying separate client algorithms to both illumination directions



In `3d` mode, the device performs the combination internally and delivers one processed result. During multiframe operation, this calculation is completed before the frame becomes available to the client.

13. Best practices

- ✓ **Prefer multiframe mode for repeated acquisition.** It separates acquisition timing from HTTP transfer and provides the intended high-throughput workflow.
- ✓ **Use direct `scan` for a genuinely isolated image.** It keeps a one-shot implementation simple.
- ✓ **Configure before starting.** Apply resolution, color, illumination, format, and virtual-rescan settings before entering multiframe mode.
- ✓ **Prefer JPEG whenever the application permits it.** Set `fileformat:jpeg` when lossy compression is acceptable. The device's in-house JPEG compression is optimized for the highest throughput while maintaining the best possible image quality.
- ✓ **Disable sharpening for maximum throughput.** Set `sharpen:-7` to bypass the sharpening processing stage completely. Values from `-6` through `7` enable this processing stage and add processing overhead.
- ✓ **Use one logical queue consumer per session.** Serialize reads and make ownership of `multiframe:next` explicit to avoid skipped or duplicated frames.
- ✓ **Minimize repeated access to the same frame.** For the highest multiframe throughput, retrieve each queued frame as few times as practical—ideally with one `image` or `preview` request followed by `multiframe:next`. Multiple ROI requests are supported, but every request adds image-processing, encoding, and HTTP overhead while the read pointer remains on the current frame. When appropriate, retrieve the complete frame once and extract ROIs in the client; use server-side ROIs when the reduction in transferred image data outweighs the additional requests.
- ✓ **Advance only after all derivatives are retrieved.** Read every required ROI or preview from the current frame, then add `multiframe:next` to the last request.
- ✓ **Consume the queue continuously.** Design for back-pressure and application stalls because the internal image queue is finite.
- ✓ **Stream binary responses.** Avoid buffering large device images more than necessary, and observe the declared content type and length.
- ✓ **Apply explicit timeouts.** Acquisition and transfer time depend on resolution, image dimensions, processing, and network throughput; use separate connection and operation timeouts.
- ✓ **Do not cache API responses.** Each request represents current device state or image data.
- ✓ **Shut down cleanly.** Stop and exit multiframe mode, then close the session, including on cancellation and recoverable errors.

Recommended client architecture: keep session ownership, acquisition state, and queue advancement in one component. Pass completed image payloads to processing workers after retrieval rather than allowing several workers to manipulate the device queue independently.

14. Error handling

Do not rely on the HTTP status alone to determine whether a device function succeeded. Parse textual S2N responses and check the response content type before treating a payload as image data.

Code	Meaning	Client response
01	Device or session busy	Back off; determine whether another client owns the device.
02	Invalid session	Discard local session state and open a new session if appropriate.
10	Invalid setting value	Query the supported range or enumeration and correct the request.
11	Unknown or unavailable setting	Check device capabilities and spelling.
30	Unsupported file format	Select a format reported by the device.
33	No image available	Wait for acquisition or review the multiframe state.
34	Width incompatible with requested rotation	Adjust the ROI, width, or rotation setting.
99	Internal error	Record the complete response and device state; recover the session where possible.

A production client should retain the numeric code and complete diagnostic text in logs. Avoid an unlimited immediate retry loop: configuration errors require correction, while busy and not-yet-available conditions may justify bounded retry with delay.

15. Quick reference

The request targets below use a direct HTTP connection to `192.168.1.50` on port 80. Send `Host: 192.168.1.50` with every HTTP/1.1 request.

Function	Session?	Purpose	Example request target
<code>s2ninfo</code>	No	Device information and capabilities	<code>/cgi/s2ninfo</code>
<code>s2nopen</code>	No	Open exclusive session	<code>/cgi/s2nopen</code>
<code>s2nclose</code>	Yes	Close session	<code>/cgi/s2nclose?SESSION</code>
<code>get</code>	Yes	Read settings or supported values	<code>/cgi/get?SESSION+SETTING</code>
<code>set</code>	Yes	Change settings	<code>/cgi/set?SESSION+SETTING:VALUE</code>
<code>scan</code>	Yes	Single acquisition	<code>/cgi/scan?SESSION</code>
<code>scan</code>	Yes	Start multiframe	<code>/cgi/scan?SESSION+multiframe:start</code>
<code>scan</code>	Yes	Stop multiframe acquisition	<code>/cgi/scan?SESSION+multiframe:stop</code>
<code>scan</code>	Yes	Exit multiframe mode	<code>/cgi/scan?SESSION+multiframe:exit</code>
<code>image</code>	Yes	Retrieve image; retain current frame	<code>/cgi/image?SESSION+...</code>
<code>image</code>	Yes	Retrieve image; then advance	<code>/cgi/image?SESSION+...+multiframe:next</code>
<code>preview</code>	Yes	Retrieve preview, optionally advance	<code>/cgi/preview?SESSION+...</code>

Multiframe state summary

Command	Acquisition after command	Scan mode after command	Queued retrieval
<code>multiframe:start</code>	Running	Multiframe	Frames are queued
<code>multiframe:stop</code>	Stopped	Multiframe	Acquisition stops; mode is retained
<code>multiframe:exit</code>	Stopped	Direct	Return to single-scan operation

Most important queue rule: an image or preview request without `multiframe:next` does not advance the read pointer. A request with `multiframe:next` returns the current result and then advances.