

皆さん、こんにちは







簡煒航 Jian Weihang

@tonytonyjan



- Founder of brainana.com



- Co-Founder of 5xruby.tw



- Coach of Rails Girls Taipei

- Champion in Yahoo Hack Day Taiwan 2013

- 4 Years Java Experience
- 2 Years Ruby Experience
- Double Keyboard Player



台灣 (Taiwan)



Ruby Meets Sony Camera Remote API

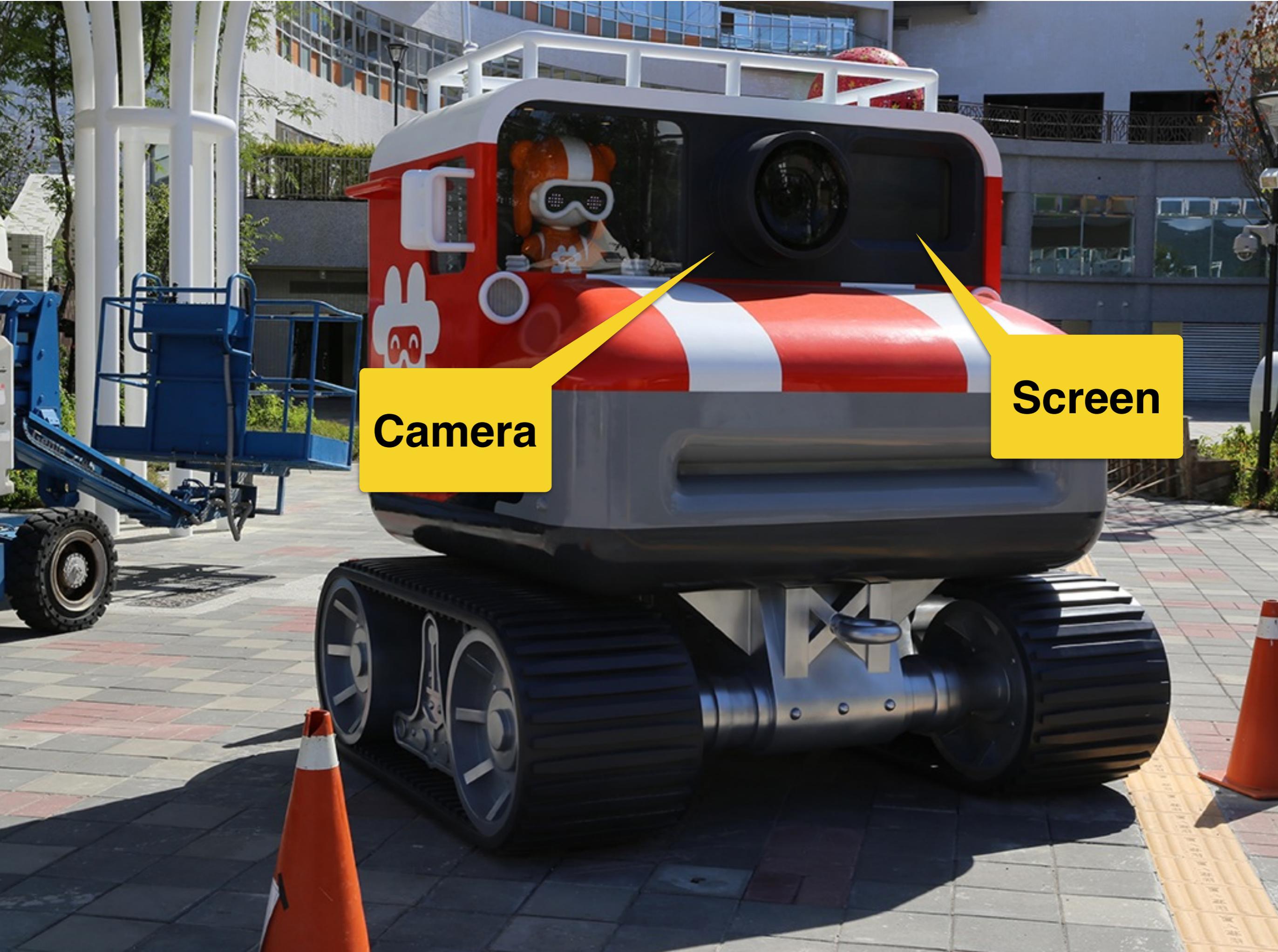
API Wrapper Implementation, and Stream Processing

3 months ago...

Taipei City New Recreation Center



Photo Truck



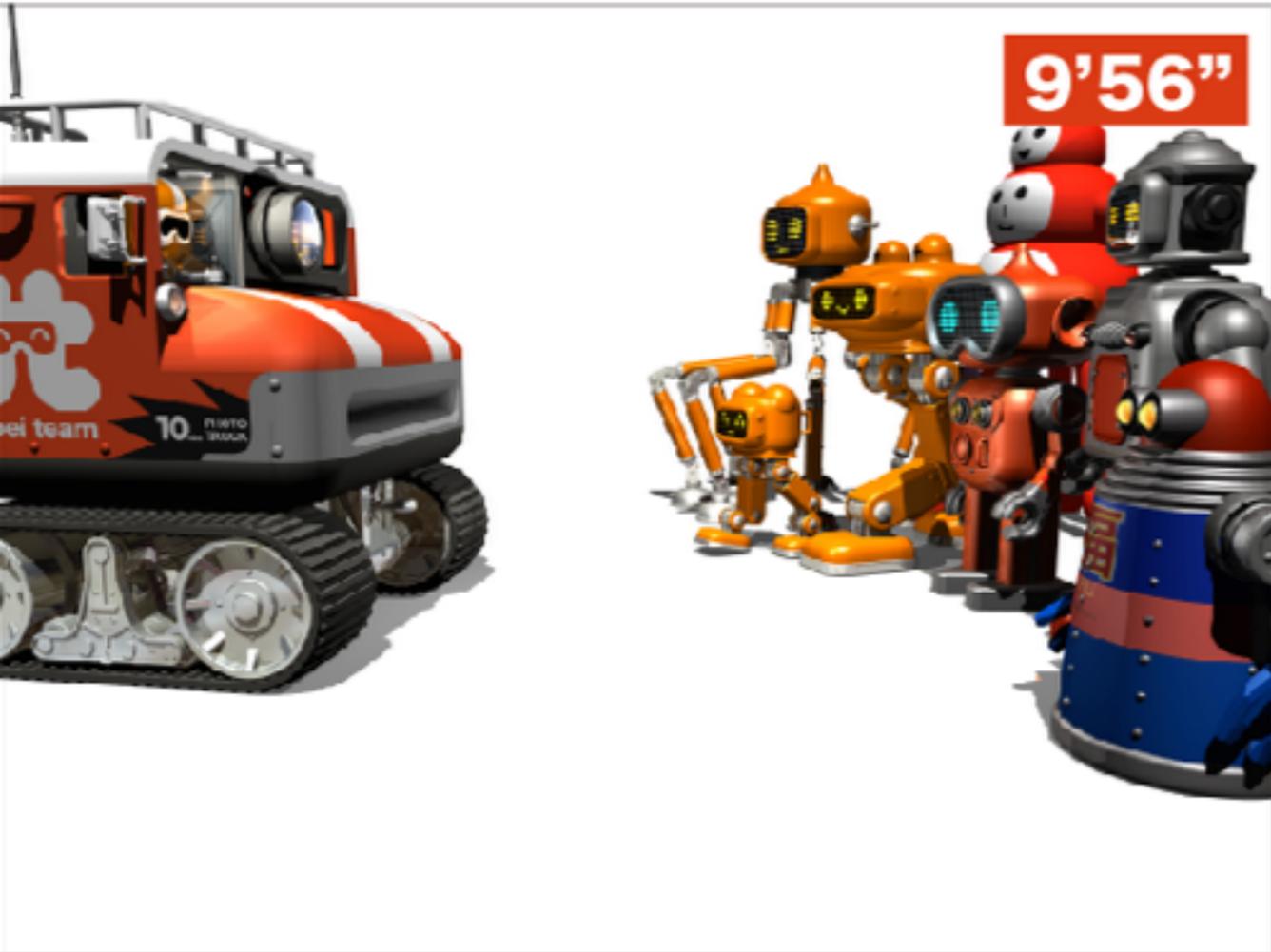
Camera

Screen

Flow

1. Play a short film.
2. Display camera live preview.
3. Take a picture.
4. Freeze the picture for 5 min
5. Repeat.





5'50"



Solutions

Solution to Display

- How about VLC API?
 - Easy to control over TCP (gem install vli-client)
 - Impossible to add effects (Countdown images, sounds)
- HTML5 over browser seems the first choice.

Solution to Camera Control

- How about gphoto2?
 - Supports more than 1,800 cameras.
 - There is CLI mode.
 - Have to repeat capturing preview to stream.
 - It's surprising.

Sony Remote Camera



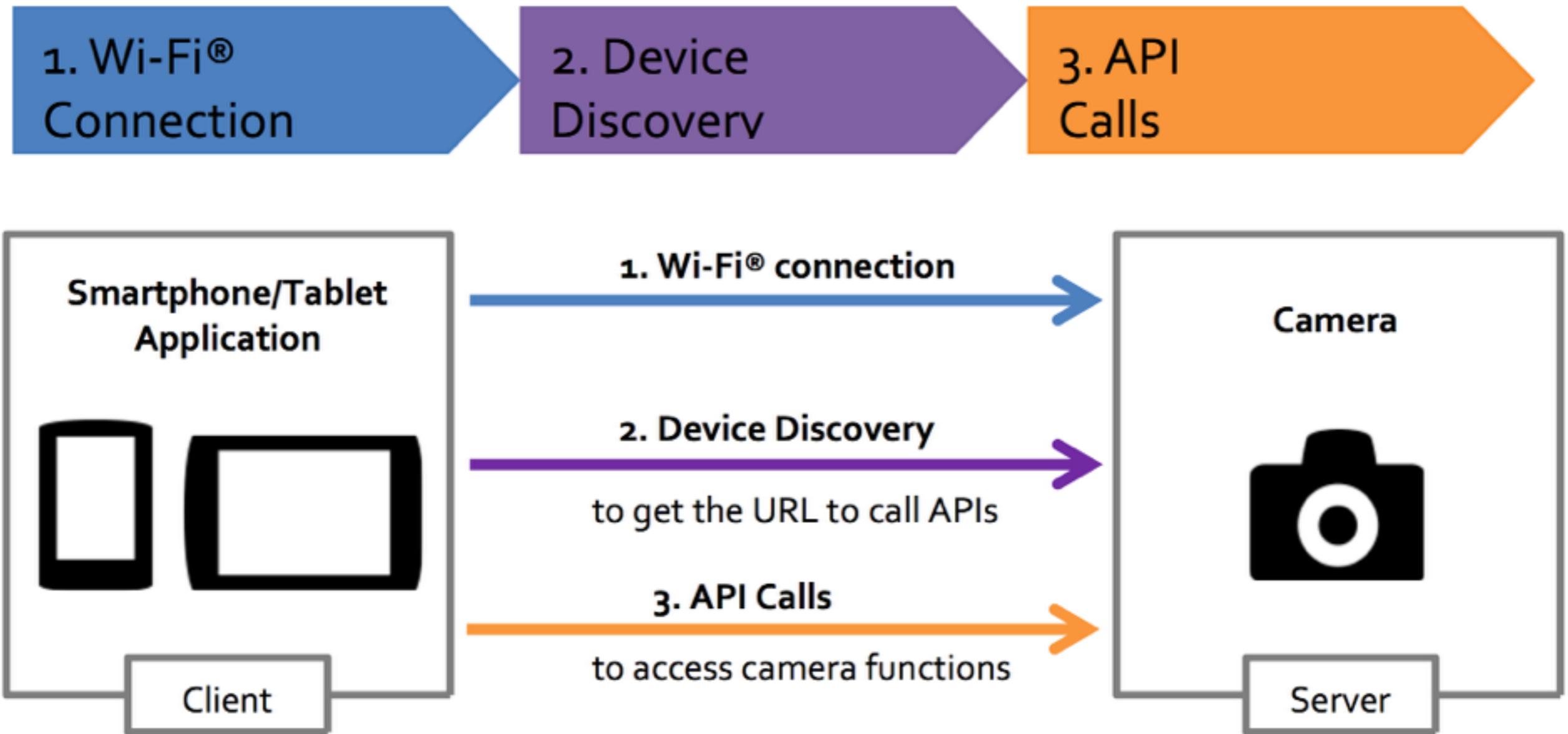
DSC-QX100

Sony Camera Remote API

- It's free, and it's open.
- It's SSDP + UPnP over Wi-Fi, and it's open.
- It's easy (JSON-RPC over HTTP), and it's open.
- It's well documented, and it's open.

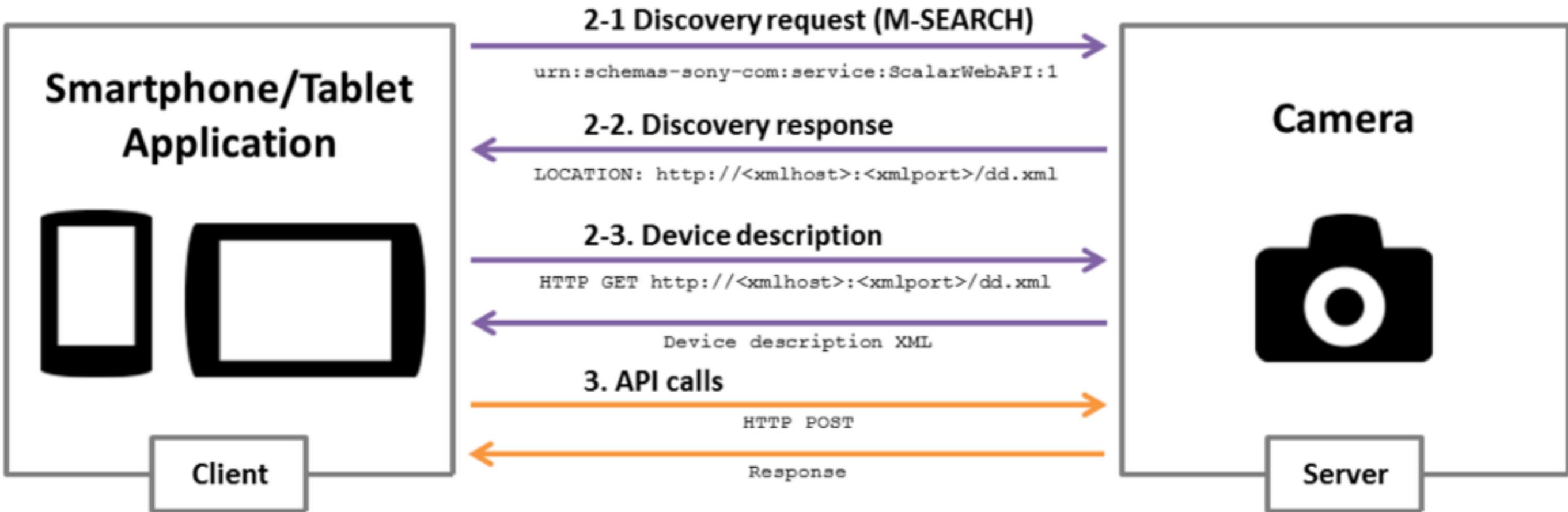
大事なことなので 4 回言いました

3 Steps to Access Camera



Device Discovery

Get the API URL



SSDP

Request

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: 10
ST: urn:schemas-sony-com:service:ScalarWebAPI:1
```

Response

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=1800
EXT:
LOCATION: http://10.0.0.1:64321/DmsRmtDesc.xml
SERVER: UPnP/1.0 SonyImagingDevice/1.0
ST: urn:schemas-sony-com:service:ScalarWebAPI:1
USN: uuid:00000000-0005-0010-8000-1c994c993998::urn:schemas-sony-com:service:ScalarWebAPI:1
X-AV-Physical-Unit-Info: pa=""; pl=;
X-AV-Server-Info: av=5.0; hn=""; cn="Sony Corporation"; mn="SonyImagingDevice"; mv="1.0";
```

Get API URL

```
<av:X_ScalarWebAPI_Service>  
  <av:X_ScalarWebAPI_ServiceType>camera</av:X_ScalarWebAPI_ServiceType>  
  <av:X_ScalarWebAPI_ActionList_URL>http://10.0.0.1:10000/sony</av:X_ScalarWebAPI_ActionList_URL>  
</av:X_ScalarWebAPI_Service>
```

URL: <http://10.0.0.1:10000/sony/camera>

API Examples

There are more than 90 APIs

JSON-RPC 1.0

Request

```
{  
  "method": "echo",  
  "params": ["Hello JSON-RPC"],  
  "id": 1  
}
```

Response

```
{  
  "result": "Hello JSON-RPC",  
  "error": null,  
  "id": 1  
}
```

Camera Remote API uses JSON-RPC over HTTP POST request.

Take Picture

Request

```
{  
  "method": "actTakePicture",  
  "params": [],  
  "id": 1,  
  "version": "1.0"  
}
```

Response

```
{  
  "result": [  
    ["http://ip:port/postview/postview.jpg"]  
  ],  
  "id": 1  
}
```

Zoom in

Request

```
{  
  "method": "actZoom",  
  "params": ["in", "start"],  
  "id": 1,  
  "version": "1.0"  
}
```

Response

```
{  
  "result": [0],  
  "id": 1  
}
```

Set Exposure

Request

```
{  
  "method": "setExposureMode",  
  "params": ["Intelligent Auto"],  
  "id": 1,  
  "version": "1.0"  
}
```

Response

```
{  
  "result": [0],  
  "id": 1  
}
```

Available Modes

“Program Auto”, “Aperture, Shutter”,
“Manual”, “Intelligent Auto”, “Superior Auto”

Ruby Time

Integrate Remote API with Ruby.

Discover Device - 1/2

```
m_search = <<-EOS  
M-SEARCH * HTTP/1.1\r  
HOST: 239.255.255.250:1900\r  
MAN: "ssdp:discover"\r  
MX: 10\r  
ST: urn:schemas-sony-com:service:ScalarWebAPI:1\r  
\r  
EOS
```

Discover Device - 2/2

```
require 'socket'
sock = UDPSocket.new
sock.bind('10.0.1.1', 0)
sock.send(m_search, 0, '239.255.255.250', 1900)
sock.recv(1024)
# =>
# HTTP/1.1 200 OK
# ...
# LOCATION: http://10.0.0.1:64321/DmsRmtDesc.xml
# ...
```

Parse XML to get API URL (using nokogiri or rexml).

Calling API

```
json = {  
  method: 'actZoom',  
  params: ['in', 'start'],  
  id: 1,  
  version: '1.0'  
}.to_json  
  
Net::HTTP.start(host, port){  
  http.request_post(path, json).body  
}
```

Live Preview

Get Liveview URL

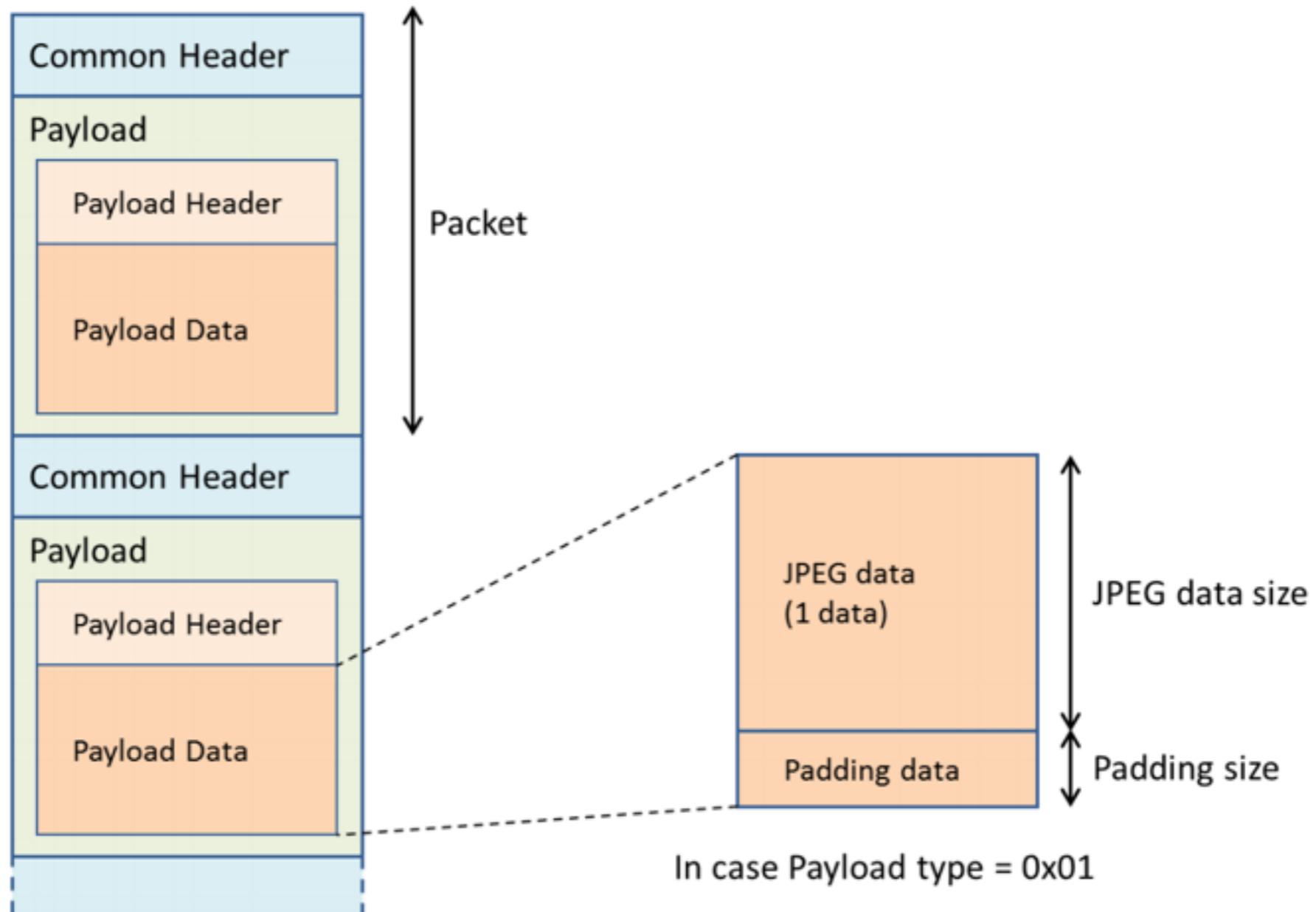
Request

```
{  
  "method": "startLiveview",  
  "params": [],  
  "id": 1,  
  "version": "1.0"  
}
```

Response

```
{  
  "result": [  
    "http://ip:port/liveview/liveviewstream"  
  ],  
  "id": 1  
}
```

Packet Data Format



Packet Format

- Comen Header: 8 bytes
- Payload Header: 128 bytes
 - Fist 4 bytes are fixed start code:
"\x24\x35\x68\x79"
 - The following 3 bytes is JPEG data size.
- Payload data: depends on JPEG data size.

Ruby Time

Processing Stream using Ruby

HTTP Streaming (1/2)

```
1 Net::HTTP.start(uri.host, uri.port) do |http|
2   request = Net::HTTP::Get.new uri
3   http.request request do |response|
4     response.read_body do |chunk|
5       # ...
6     end
7   end
8 end
```

HTTP Streaming (2/2)

```
1 Net::HTTP.start(uri.host, uri.port) do |http|
2   request = Net::HTTP::Get.new uri
3   http.request request do |response|
4     response.read_body do |chunk|
5       buf += chunk
6       until buf.empty?
7         # buf.slice!
8       end
9     end
10  end
11 end
```

String#unpack

Decodes string, returning array of each value extracted.

Common Header

Common Header (1/2)

0	1	2	3
Start Byte	Payload Type	Sequence Number	
4	5	6	7
Time Stamp			

```
ary = common_header.unpack("aanN")
```

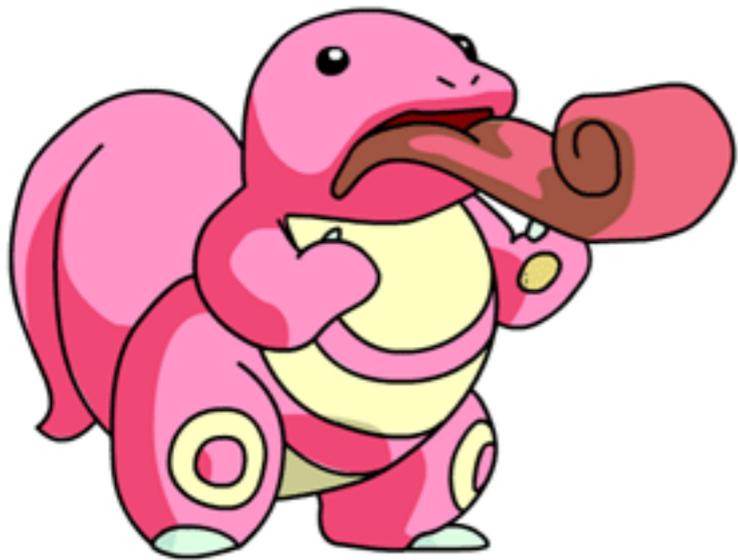
```
ary[2] # => Sequence Number
```

```
ary[3] # => Timestamp
```

Common Header (2/2)

	return	meaning
--	-----	-----
a	String	arbitrary binary string
n	Integer	16-bit unsigned, big-endian
N	Integer	32-bit unsigned, big-endian

Payload Header



Payload Header (1/2)

0	1	2	3
Start Code			
5	6	7	8
JPEG Data Size			Padding Size
9	10	11	12
Reserved			
13	14	...	127
Flag	Reserved		

```
ary = payload_header.unpack('a4H6Ca*')
```

```
ary[1].hex # => JPEG Size
```

```
ary[2]     # => Padding Size
```

Payload Header (2/2)

	return	meaning
--	-----	-----
C	Integer	8-bit unsigned (unsigned char)
H	String	hex string (high nibble first)
h	String	hex string (low nibble first)

Ruby gem?

```
$ gem install sonycam
```

<https://github.com/tonytonyjan/sonycam>

Ruby Usage

```
require 'sonycam'  
api = Sonycam::API.new "http://10.0.0.1:10000/sony/camera"  
api.request :actTakePicture  
# => [["http://....."]]  
api.request :actZoom, :in, :start  
# => 0
```

```
Liveview.stream(liveview_url) do |packet|  
  packet[:payload_data][:jpeg_data] # JPEG binary  
end
```

CLI Usage

```
$ gem install sonycam
```

```
$ sonycam scan
```

```
$ sonycam api actTakePicture
```

CLI Usage

```
~ $ sonycam help
Commands:
sonycam api method [PARAMETER ...]
sonycam help [COMMAND]
sonycam list [QUERY]
sonycam liveview
sonycam scan [IP]
```

`sonycam liveview` prints streaming data to STDOUT

Record to mp4

```
$ sonycam liveview | ffmpeg \  
-f image2pipe -c mjpeg \  
-i pipe:0 -codec copy \  
liveview.mp4
```

Live Streaming

```
$ sonycam liveview | ffmpeg \  
-f image2pipe -c mjpeg \  
-i pipe:0 -codec copy \  
http://127.0.0.1:8080/feed1.ffm
```

Friendly Reminder

Secrets in DSC-RX100M2

- Others
 - <http://10.0.0.1:10000/sony/camera>
- DSC-RX 100M2
 - <http://10.0.0.1:10000/camera>



It's not mentioned in any official document.

Mandatory Extensions (1/2)

```
M-SEARCH * HTTP/1.1
```

```
HOST: 239.255.255.250:1900
```

```
MAN: "ssdp:discover"
```

```
MX: 10
```

```
ST: urn:schemas-sony-com:service:ScalarWebAPI:1
```

Mandatory Extensions (2/2)

MAN

REQUIRED by HTTP Extension Framework. Unlike the NTS and ST field values, the field value of the MAN header field is enclosed in double quotes; it defines the scope (namespace) of the extension.

MUST be **“ssdp:discover”**.

- *Quoted from “UPnP Device Architecture 1.1”*

Conclusion

Sony's Cameras are
friendly for developers

Ruby is easy to write
even in handling binary

Thank You

```
"tonytonyjan".is_a? Singleton # => true
```