皆さん、こんにちちは
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
Flow

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

1. Wi-Fi® Connection
2. Device Discovery
3. API Calls

- Smartphone/Tablet Application
  - 1. Wi-Fi® connection
  - 2. Device Discovery
    - to get the URL to call APIs
  - 3. API Calls
    - to access camera functions

- Camera
- Client
- Server
Device Discovery
Get the API URL

1. Discovery request (M-SEARCH)
   urn:schemas-sony-com:service:ScalarWebAPI:1

2. Discovery response
   LOCATION: http://<xmlhost>:<xmlport>/dd.xml

3. Device description
   HTTP GET http://<xmlhost>:<xmlport>/dd.xml
   Device description XML

4. API calls
   HTTP POST
   Response
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

URL:  http://10.0.0.1:10000/sony/camera
API Examples

There are more than 90 APIs
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

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

Response

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

Available Modes

Request

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

Response

```
{
    "result": [0],
    "id": 1
}
```
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\rEOS
Discover Device - 2/2

```ruby
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

```javascript
json = {
  method: 'actZoom',
  params: ['in', 'start'],
  id: 1,
  version: '1.0'
}.to_json

Net::HTTP.start(host, port){
  http.request_post(path, json).body
}
```
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

Common Header
Payload
  Payload Header
  Payload Data

Common Header
Payload
  Payload Header
  Payload Data

JPEG data
  (1 data)
  Padding data

In case Payload type = 0x01

JPEG data size
Padding size
Packet Format

- Comen Header: 8 bytes
- Payload Header: 128 bytes
  - First 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)

```ruby
Net::HTTP.start(uri.host, uri.port) do |http|
  request = Net::HTTP::Get.new uri
  http.request request do |response|
    response.read_body do |chunk|
      # ...
    end
  end
end
```
HTTP Streaming (2/2)

```ruby
Net::HTTP.start(uri.host, uri.port) do |http|
  request = Net::HTTP::Get.new uri
  http.request request do |response|
    response.read_body do |chunk|
      buf += chunk
      until buf.empty?
        # buf.slice!
      end
    end
  end
end
```
String#unpack

Decodes string, returning array of each value extracted.
Common Header (1/2)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Byte</td>
<td>Payload Type</td>
<td>Sequence Number</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time Stamp</td>
</tr>
</tbody>
</table>

```python
ary = common_header.unpack("aanN")
ary[2]  # => Sequence Number
ary[3]  # => Timestamp
```
<table>
<thead>
<tr>
<th>return</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>String</td>
</tr>
<tr>
<td>n</td>
<td>Integer</td>
</tr>
<tr>
<td>N</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Payload Header
Payload Header (1/2)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Start Code</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>JPEG Data Size</strong></td>
<td><strong>Padding Size</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td><strong>Reserved</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td></td>
<td>127</td>
<td></td>
</tr>
<tr>
<td><strong>Flag</strong></td>
<td><strong>Reserved</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
ary = payload_header.unpack(\'a4H6Ca*\')
ary[1].hex  # => JPEG Size
ary[2]      # => Padding Size
```
<table>
<thead>
<tr>
<th>return</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Integer</td>
</tr>
<tr>
<td>H</td>
<td>String</td>
</tr>
<tr>
<td>h</td>
<td>String</td>
</tr>
</tbody>
</table>

- Integer: 8-bit unsigned (unsigned char)
- String: hex string (high nibble first)
- String: hex string (low nibble first)
Ruby gem?
$ gem install sonycam

https://github.com/tonytonyjan/sonycam
Ruby Usage

```ruby
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

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

```bash
$ sonyacam liveview | ffmpeg \
-f image2pipe -c mjjpeg \
-i pipe:0 -codec copy \nhttp://127.0.0.1:8080/feed1.ffmpeg
```
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