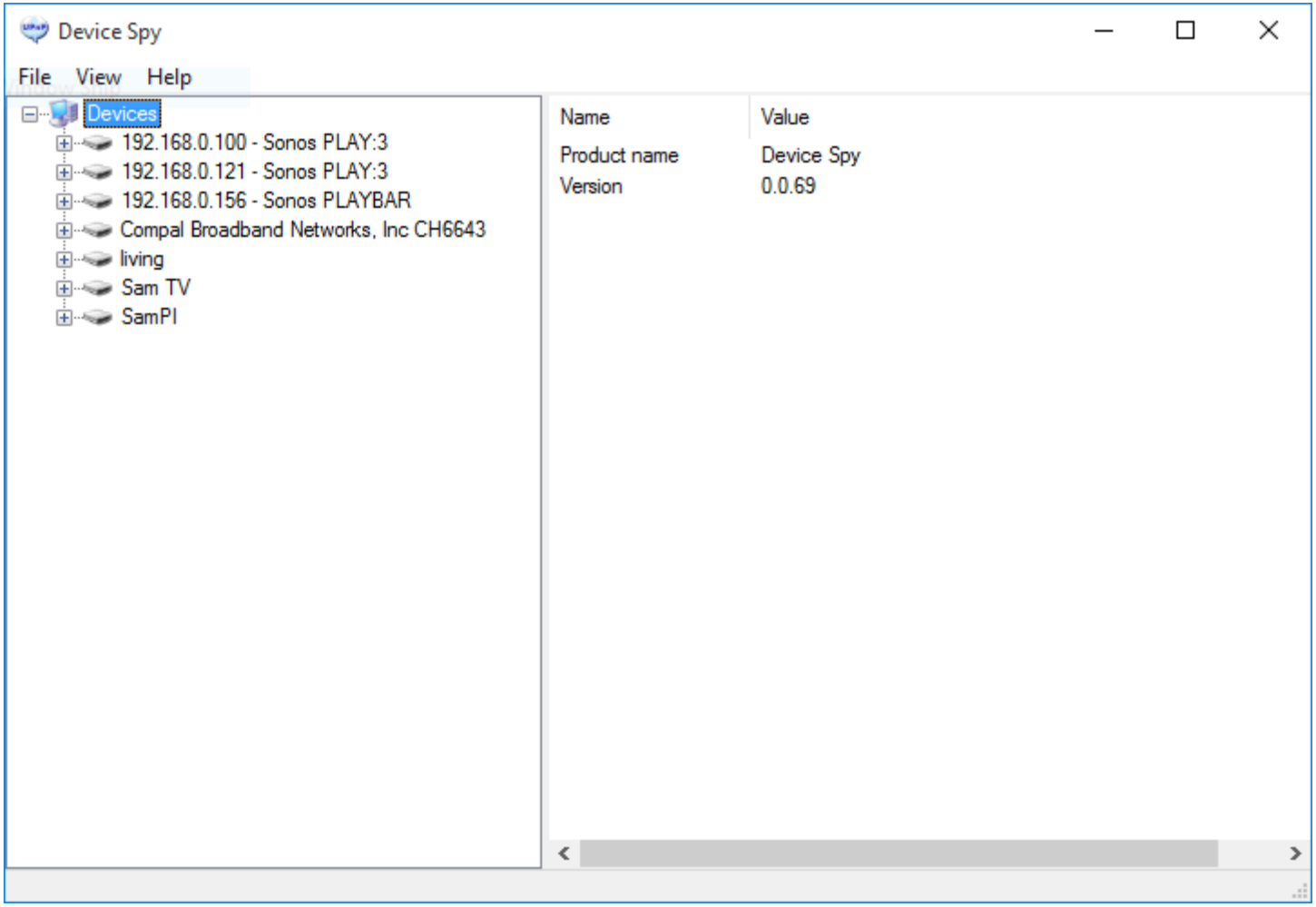
A look behind the scenes

Posted by Sam Neirinck on July 25, 2015

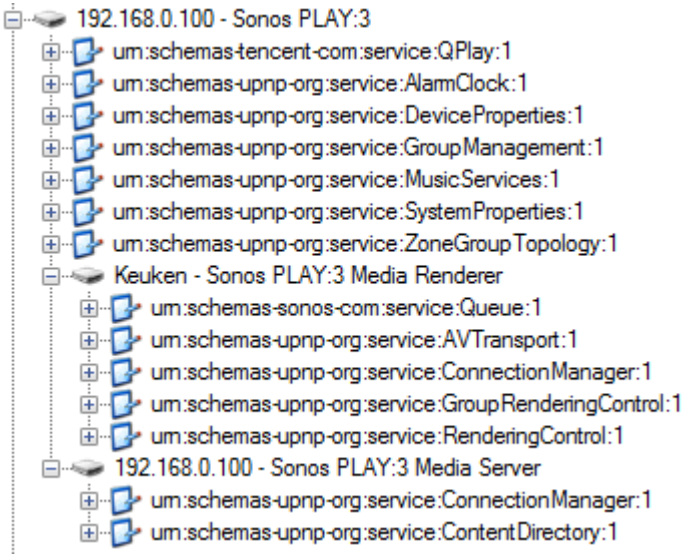
While Sonos has no official API, it uses open technologies for the apps to communicate with the devices, namely Universal Plug and Play, UPnP. It's highly likely that you have a device at home supporting UPnP. Things like routers, TV's and set-top boxes all have UPnP services.

A device supporting UPnP is discoverable and can announce which services it support, along with which functions clients can call and what arguments the device expects. There's a great utility on Windows to see all this information: Device Spy. It's included in the Developer tools for UPnP (http://opentools.homeip.net/dev-tools-for-upnp).

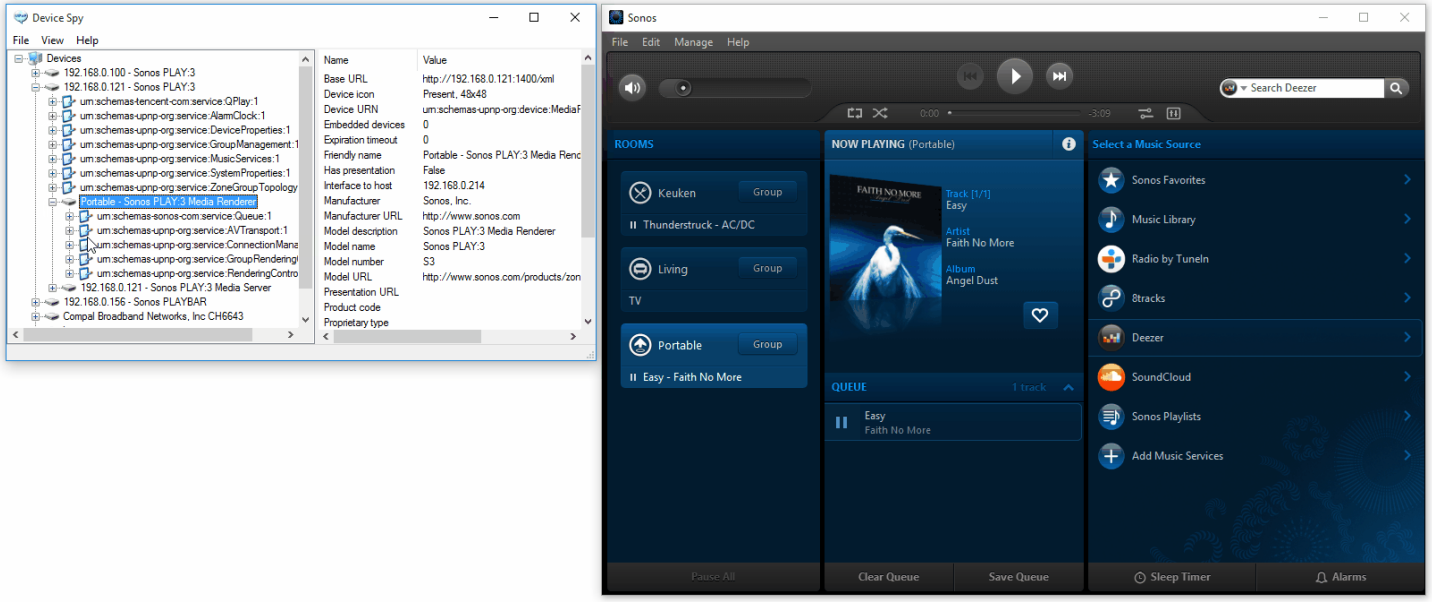
On my network, it finds the following devices:



You can see the 3 Sonos devices, a router, a TV, a Raspberry Pi, and my set-top box. If I expand the Play 3, we can see it exposes quite a few services.



With Device Spy we can invoke a function, and the Sonos controller application immediately responds to this.



(/img/device-spy-sonos.gif)

Behind the scenes, the invocation of this method is done via SOAP. Getting the volume of a speaker is done like this:

```
POST /MediaRenderer/RenderingControl/Control HTTP/1.1
HOST: 192.168.0.156:1400
SOAPACTION: "urn:schemas-upnp-org:service:RenderingControl:1#GetVolume"
CONTENT-TYPE: text/xml; charset="utf-8"
Content-Length: 380

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:s="http://schemas.xmlsoa
   <s:Body>
      <u:GetVolume xmlns:u="urn:schemas-upnp-org:service:RenderingControl:1">
         <InstanceID>0</InstanceID>
         <Channel>Master</Channel>
      </u:GetVolume>
   </s:Body>
</s:Envelope>
```

w raw
ps://gist.github.com/samneirinck/904b55b2b75c8f0370cf/raw/a246c829c4c9b6ac377bed52aad34ad636d7e6e7/request.xml)
request.xml (https://gist.github.com/samneirinck/904b55b2b75c8f0370cf#file-request-xml) hosted with ♥ by **GitHub (https://github.com)**

The speaker returns with a straight-forward reply:

```
HTTP/1.1 200  OK
EXT:
CONTENT-TYPE:  text/xml; charset="utf-8"
SERVER:  Linux UPnP/1.0 Sonos/29.5-91030 (ZPS9)
CONNECTION:  close
```

```
Content-Length: 288

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle="http://schemas.xmlsoa
        <s:Body>
                <u:GetVolumeResponse xmlns:u="urn:schemas-upnp-org:service:RenderingControl:1">
                        <CurrentVolume>15</CurrentVolume>
                </u:GetVolumeResponse>
        </s:Body>
</s:Envelope>
```

v raw
ps://gist.github.com/samneirinck/8b895b7c8e9061818fef/raw/ddcbbe0aab4e3f803d6fe05227459179ffc0991e/response.xml)
response.xml (https://gist.github.com/samneirinck/8b895b7c8e9061818fef#file-response-xml) hosted with ♥ by **GitHub (https://github.com)**

Although this will get you along the road, we still have a missing piece. The apps from Sonos react immediately to any change (volume, next track, mute, etc…). This is done by using UPnP Eventing. In essence, each controller hosts a simple http server, and then notifies the speaker that it wants to subscribe to events. Whenever an event happens, the speaker will notify all of its subscribers of the new value.

We'll want this functionality in SonosSharp as well, and abstract the HTTP server away, since it has to be cross-platform.

In the next blog post we'll dive into the code, or at least discuss on how we should structure it.

← PREVIOUS POST (/2015/06/20/SONOSSHARP-A-NEW-BEGINNING/)

NEXT POST → (/2016/08/01/EASY-JEKYLL-BLOGGING-ON-WINDOWS-USING-DOCKER/)

(/feed.xml) (https://twitter.com/samneirinck)

(https://github.com/samneirinck)