LAN Devices Quick Start

The starter projects demonstrate the best practices to create an efficient LAN device integration.

Exploring the Quick Start Project Demo:

Service Manager:
https://github.com/juano2310/SmartThings_templates/blob/master/samsung_connect.groovy

Device Type:
https://github.com/juano2310/SmartThings_templates/blob/master/samsung_dt.groovy

Step by Step:

1 - Discover devices over LAN
2 - Add selected devices as children
3 - Define Device Specific functions ()

1 - Discover devices over LAN

First you have to identify the UPNP device type. Personally I use Net Analyzer (https://itunes.apple.com/us/app/network-analyzer-ping-traceroute/id557405467?mt=8)

Replace with newly discovered device type

```
return "urn:samsung.com:device:RemoteControlReceiver:1" //Samsung TV
```

Send Hub Command to start discovery

```
private discoverDevices() {
	sendHubCommand(new physicalgraph.device.HubAction("lan discovery ${getDeviceType()}",
physicalgraph.device.Protocol.LAN))
}
```

Parse response, make sure that the device type match the response and store it in the device list.

```
def locationHandler(evt) {
	def description = evt.description
  log.trace "Location: $description"

	def hub = evt?.hubId
	def parsedLanEvent = parseLanMessage(description, true)
	parsedLanEvent << ["hub":hub]
```

```
	if (parsedLanEvent?.ssdpTerm?.contains(getDeviceType())) {
	//SSDP DISCOVERY EVENTS
		log.trace "Device found"
		def devices = getDevice()
		if (!(devices."${parsedLanEvent.ssdpUSN.toString()}")) {
	//device does not exist
	log.trace "Adding Device to state..."
			devices << ["${parsedLanEvent.ssdpUSN.toString()}":parsedLanEvent]
		} else {
	// update the values
			log.trace "Device was already found in state..."
			def d = devices."${parsedLanEvent.ssdpUSN.toString()}"
			if(d.ip != parsedLanEvent.networkAddress || d.port != parsedLanEvent.deviceAddress) {
				d.ip = parsedLanEvent.networkAddress
				log.trace "Device's port or ip changed..."
			}
		}
	}
	else if (parsedLanEvent.headers && parsedLanEvent.body) {
	// device RESPONSES
		def type = parsedLanEvent.headers."content-type"
		def body
		log.trace "REPONSE TYPE: $type"
		if (type?.contains("xml")) {
	// description response (application/xml)
			body = new XmlSlurper().parseText(parsedLanEvent.body)
	def devicet = getDeviceType().toLowerCase()
	def devicetxml = body.device.deviceType.text().toLowerCase()
	log.trace "$devicetxml == $devicet"
			if (devicetxml == devicet) {
				def devices = getDevice()
				def device = devices.find {it?.key?.contains(body?.device?.UDN?.text())}
				if (device) {
					device.value <<
[name:body?.device?.friendlyName?.text(),model:body?.device?.modelName?.text(),
serialNumber:body?.device?.serialNum?.text(), verified: true]
				} else {
					log.error "The xml file returned a device that didn't exist"
				}
			}
		}
		else if(type?.contains("json")) {
	//(application/json)
			body = new groovy.json.JsonSlurper().parseText(bodyString)
			log.trace "GOT JSON $body"
		}
	}
	else {
		log.trace "Device not found..."
```

```
                //log.trace description
        }
}

Map devicesDiscovered() {
        def vdevices = getVerifiedDevice()
        def map = [:]
    log.trace "Discovered $vdevices"
        vdevices.each {
                def value = "${it.value.name}"
                def key = it.value.mac
                map["${key}"] = value
        }
        map
}
```

2 - Add selected devices as children

```
//CHILD DEVICE METHODS
def addDevice(){
        def devices = getVerifiedDevice()
    log.trace "Adding childs"
        selecteddevice.each { dni ->
                def d = getChildDevice(dni)
                if(!d) {
                        def newDevice = devices.find { (it.value.mac) == dni }
          def deviceName = getDeviceName() + "[${newDevice?.value.name}]"
                        d = addChildDevice(getNameSpace(), getDeviceName(), dni, newDevice?.value.hub,
[label:"${deviceName}"])
                        log.trace "Created ${d.displayName} with id $dni"
                } else {
                        log.trace "${d.displayName} with id $dni already exists"
                }
        }
}
```

3 - Define Device Specific functions ()

```
private tvAction(key,deviceNetworkId) {
    log.debug "Executing ${key}"

        def tvs = getVerifiedDevice()
        def thetv = tvs.find { (it.value.mac) == deviceNetworkId }

    // Standard Connection Data
    def appString = "iphone..iapp.samsung"
    def appStringLength = appString.getBytes().size()

    def tvAppString = "iphone.UN60ES8000.iapp.samsung"
    def tvAppStringLength = tvAppString.getBytes().size()
```

```
    def remoteName = "SmartThings".encodeAsBase64().toString()
    def remoteNameLength = remoteName.getBytes().size()

    // Device Connection Data
    def ipAddress = convertHexToIP(thetv?.value.networkAddress).encodeAsBase64().toString()
    def ipAddressHex = thetv?.value.networkAddress
    def ipAddressLength = ipAddress.getBytes().size()

    def macAddress = thetv?.value.mac.encodeAsBase64().toString()
    def macAddressLength = macAddress.getBytes().size()

    // The Authentication Message
    def authenticationMessage =
"${(char)0x64}${(char)0x00}${(char)ipAddressLength}${(char)0x00}${ipAddress}${(char)macAddressLength}${(char)0
x00}${macAddress}${(char)remoteNameLength}${(char)0x00}${remoteName}"
    def authenticationMessageLength = authenticationMessage.getBytes().size()

    def authenticationPacket =
"${(char)0x00}${(char)appStringLength}${(char)0x00}${appString}${(char)authenticationMessageLength}${(char)0x00
}${authenticationMessage}"

    // If our initial run, just send the authentication packet so the prompt appears on screen
    if (key == "AUTHENTICATE") {
                sendHubCommand(new physicalgraph.device.HubAction(authenticationPacket,
physicalgraph.device.Protocol.LAN, "${ipAddressHex}:D6D8"))
    } else {
        // Build the command we will send to the Samsung TV
        def command = "KEY_${key}".encodeAsBase64().toString()
        def commandLength = command.getBytes().size()

        def actionMessage =
"${(char)0x00}${(char)0x00}${(char)0x00}${(char)commandLength}${(char)0x00}${command}"
        def actionMessageLength = actionMessage.getBytes().size()

        def actionPacket =
"${(char)0x00}${(char)tvAppStringLength}${(char)0x00}${tvAppString}${(char)actionMessageLength}${(char)0x00}${a
ctionMessage}"

        // Send both the authentication and action at the same time
        sendHubCommand(new physicalgraph.device.HubAction(authenticationPacket + actionPacket,
physicalgraph.device.Protocol.LAN, "${ipAddressHex}:D6D8"))
    }
}
```

Next Steps/ First Project

Now we have to install this into the IDE and test this example.

Best Practices, Considerations and FAQ

- Device network ID (DNI) HAS TO BE UNIQUE AND NEVER CHANGE
Recommendation, USE MAC ADDRESS!

- UPnP NOTIFY
NOTIFY should be coming back to dev-conn and then passed up to data-mgmt with a subscriptionId as part of the message.

For example:

```
if (headerString.contains(“SID: uuid:”)) {
 def sid = (headerString =~ /SID: uuid:.*/) ? ( headerString =~ /SID: uuid:.*/)[0] : “0”
 sid -= “SID: uuid:”.trim()

 updateDataValue(“subscriptionId”, sid)
 }
```

Wemo Motion and Switch are two examples you could look at to see how this is done.

Feedback and next steps

Community dedicated thread?