



(FP7 614100)

## **D3.2 Resource and Service Discovery Solutions**

**2015-02-28 – Version 1.0**

**Published by the IMPReSS Consortium**

**Dissemination Level: Public**



**Project co-funded by the European Commission within the 7th Framework Programme and  
the Conselho Nacional de Desenvolvimento Científico e Tecnológico  
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation**

## Document control page

**Document file:** D3.2\_Resource\_and\_Service\_Discovery\_FINAL.docx  
**Document version:** 1.0  
**Document owner:** Peter Rosengren (CNET)

**Work package:** WP 3 Resource Abstraction and IoT Communication Infrastructure  
**Task:** Task 3.2 Resource and Service Discovery  
**Deliverable type:** P

**Document status:**  approved by the document owner for internal review  
 approved for submission to the EC

### Document history:

Version	Author(s)	Date	Summary of changes made
0.3	Peeter Kool, Peter Rosengren	2015-02-01	Initial Content
0.4	Peter Rosengren	2015-02-05	Added basic concept
0.5	Matts Ahlsén, Peeter Kool, Peter Rosengren	2015-05-10	Added description of IoT Resource Catalogue
0.6	Peeter Kool	2015-05-11	Added API-description for IoT Resource
0.7	Peter Rosengren	2015-05-15	Added API-description for IoT Resource Catalogue
0.8	Peter Rosengren	2015-05-20	Description of the Catalogue Resource query language
0.9	Peeter Kool	2015-05-22	Description of IoT Resource Builder Tool
0.99	Peeter Kool, Peter Rosengren	2015-06-08	Proof-read, ready for peer review
1.0	Peeter Kool	2015-06-09	Final version after peer review

### Internal review history:

Reviewed by	Date	Summary of comments
Jussi Kiljander	2015-06-09	Minor comment on grammar.
Ferry Pramudianto	2015-06-09	Accepted

#### Legal Notice

The information in this document is subject to change without notice.

The Members of the IMPRESS Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the IMPRESS Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Possible inaccuracies of information are under the responsibility of the project. This report reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained therein.

# Index:

- 1 Executive summary ..... 4**
- 2 Introduction ..... 5**
  - 2.1 Purpose, context and scope of this deliverable ..... 5
  - 2.2 Background ..... 5
- 3 Basic Concepts ..... 6**
  - 3.1 IoTEntity ..... 6
  - 3.2 IoTResource..... 6
  - 3.3 IoTWorld ..... 6
  - 3.4 IoT Resource, Entity and Service Catalogue ..... 6
- 4 IoTResource Catalogue ..... 10**
  - 4.1 Retrieving catalogue data ..... 11
  - 4.2 IoTResource Query Language..... 14
  - 4.3 Actuating on IoTResources using the service catalogue ..... 15
  - 4.4 Enabling a service in the IMPRESS platform ..... 16
    - 4.4.1 Enabling a service on the IMPRESS network ..... 16
    - 4.4.2 Registering Service Metadata in the IoTResource Catalogue ..... 18
- 5 Development Tools ..... 21**
  - 5.1 IoTResource Builder and service annotations..... 21
    - 5.1.1 Service annotations..... 21
    - 5.1.2 Use of The resource builder ..... 21
    - 5.1.3 Components in the IoTResource Builder ..... 26
  - 5.2 IoTResource Catalogue..... 29
    - 5.2.1 Catalogue Services and Actions ..... 30
  - 5.3 IoTResource Catalogue Browser. .... 31
- 6 References ..... 33**

## 1 Executive summary

Impress provides a straightforward model for representing objects in the physical world and their properties and map them to software resources that allow control and reading of data about the physical objects. The three main concepts are IoTEntity, IoTResource and IoTWorld. The purpose of the IoT Resource and Service modules is to discover and map available functionality and services in the network to these basic concepts.

The IoTWorld is accessed through a Linksmart .net software defined gateway, called an IoTWorld Gateway. The gateway provides mechanisms for routing requests regarding the physical world to the appropriate software resource that is able to fulfil the request, such as reporting the temperature in a particular office space or the current electricity consumption of a household appliance.

This functionality is provided by the IoT Resource Catalogue. The catalogue discovers and keeps track of available IoTResources in the network. It provides a REST-based interface to select and retrieve data about IoTResources and their services. The IoT Resource Catalogue provides the means to store more elaborate metadata regarding the services compared to the Network Manager Service Catalogue. The IoT Resource Catalogue uses service descriptions that are expressed in an extended version of SCPD (Service Control Protocol Description) which is the standard for service descriptions in DLNA/UPnP.

Finally Impress also develops a number of tools that are providing automatic support for discovery. The IoTResource Builder allows a developer to define IoTResources and automatically generate the necessary IoTResource code stubs. Services can then be built using these IoTResources. The IoTResources will also automatically register themselves in both the Network Manager Service Catalogue as well as the IoTResource Catalogue.

## 2 Introduction

### 2.1 Purpose, context and scope of this deliverable

This deliverable defines the basis for discover and use of resources and services in the IMPReSS project. It is based on the current development plan and architecture. This deliverable is based on work in task 3.2 "Resource and Service Discovery". The task is responsible for defining the device and service discovery mechanisms to be used in the IMPRESS platform.

Section 3 defines basic concepts for Internet of Things discovery. Chapter 4 describes the IoT Resource catalogue which is the main component responsible for resource and service discovery. Chapter 5 describes several developer tools that provide automatic discovery support to be included into applications by developers.

### 2.2 Background

Both infrastructure-less (resources advertise) and infrastructure-based solutions (gateways advertise) is taken into consideration. The discovery functionalities are also extended to support specific search capabilities based on custom criteria. The task will address discovery aspect at both middleware level and at application domain resources level. In the last scenario, specific discovery capabilities will be analyzed and implemented within the RAI modules. Resource identification and discovery need to be done in several levels. ID abstraction for heterogeneous networks into "virtual addresses". The IoT-A reference architecture has been the starting point for defining the resource and service discovery mechanisms and tools. Discovery based on semantic matchmaking that allows resources to be found based on several parameters e.g. QoS, capabilities, classifications etc. should be integrated into LinkSmart.

CNet will lead the task and will implement resource discovery mechanisms for the specific sensors selected for the uses cases.

## 3 Basic Concepts

Impress provides a straightforward model for representing objects in the physical world and their properties and map them to software resources that allow control and reading of data about the physical objects.

The three main concepts are `IoTEntity`, `IoTResource` and `IoTWorld`. The purpose of the IoT Resource and Service modules is to discover and map available functionality and services in the network to these basic concepts.

### 3.1 IoTEntity

An `IoTEntity` is a software representation a physical entity, e.g. a house, a car or a door. `IoTEntity` corresponds to the IoT-A concept of Virtual Entity. An `IoTEntity` has properties: a house may have indoor temperature and energy consumption, a door may be locked or unlocked and opened or closed. The state of these properties at different points in time – typically generated by an `IoTResource` connected to a physical sensor – are represented by `IoTStateObservations`. The entity of interest for observations in an IoT system, the things in the physical world we want to observe and perform actions on. Examples of `IoTEntity` instances can be rooms, a household appliance or a person. The `IoTEntities` have properties that can be observed.

### 3.2 IoTResource

`IoTResources` are software objects that provide IoT Services for applications and end-users for retrieving and analysing data about the physical world as well as invoking actions like switching of a light. Some examples of `IoTResources` are software objects representing actuators, sensors, data streams, databases, etc.

### 3.3 IoTWorld

A subset of the physical world, a set of `IoTEntities` that belong together and the associated `IoTResources` for observing and acting on this part of physical world. It could represent physical, functional or organizational units in some application domain. For instance you can model your house as one `IoTWorld`, or an office, a complete building or even a whole city, it depends on the need of your Application. The `IoTWorld` is accessed through a Linksmart .net software defined gateway, called an `IoTWorld Gateway`.

### 3.4 IoT Resource, Entity and Service Catalogue

The Linksmart.net gateway provides mechanisms for routing requests regarding the physical world to the appropriate software resource that is able to fulfil the request, such as reporting the temperature in a particular office space or the current electricity consumption of a household appliance.

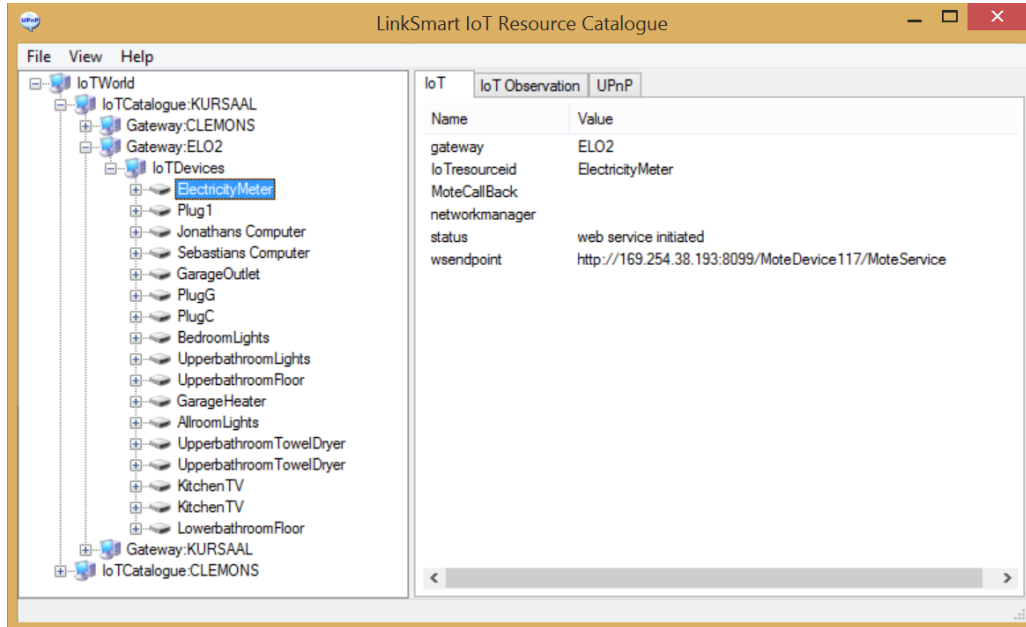
This functionality is provided by the IoT Resource Catalogue. The catalogue discovers and keeps track of available `IoTResources` in the network. It provides a REST-based interface to select and retrieve data about `IoTResources` and their services.

`IoTResources` are software objects that provide IoT Services for applications and end-users, e.g., retrieving and analysing data about the physical world, invoking actions and so on. Currently three types of `IoTResources` have been defined and implemented:

- `IoTDevice`
- `IoTSensor`
- `IoTThing`

`IoTResources` typically runs in `IoTWorld` gateways. The `IoTResources` provides the means to deliver the information about `IoTEntities` in the `IoTWorld` and to actuate on them.

As an example see figure below that shows which IoTResources have been discovered on the IoTWorld gateway "KURSAAL", which handles several physical gateways (KURSAAL, ELO2,CLEMONS) and which IoT Services they offer. IoTResources are discovered and managed by the IoT Resource Catalogue.



Each IoT Resource provides an easy-to-use REST interface to retrieve data about the resource and its current state. It is also possible to invoke actions.

### Retrieving Services

By providing the endpoint and the keyword services it is possible to retrieve all services offered.

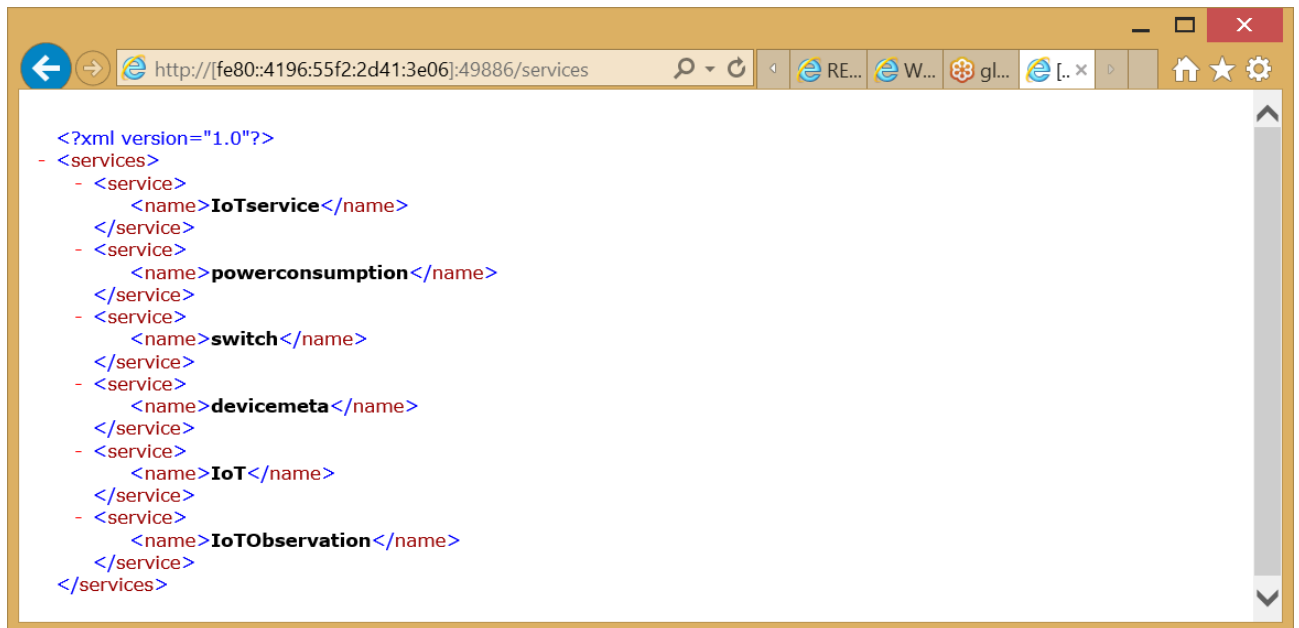


Figure 1: Service listing

### Retrieving Statevariables

Supplying the service name and the keyword statevariables will list all statevariables associated with the service.

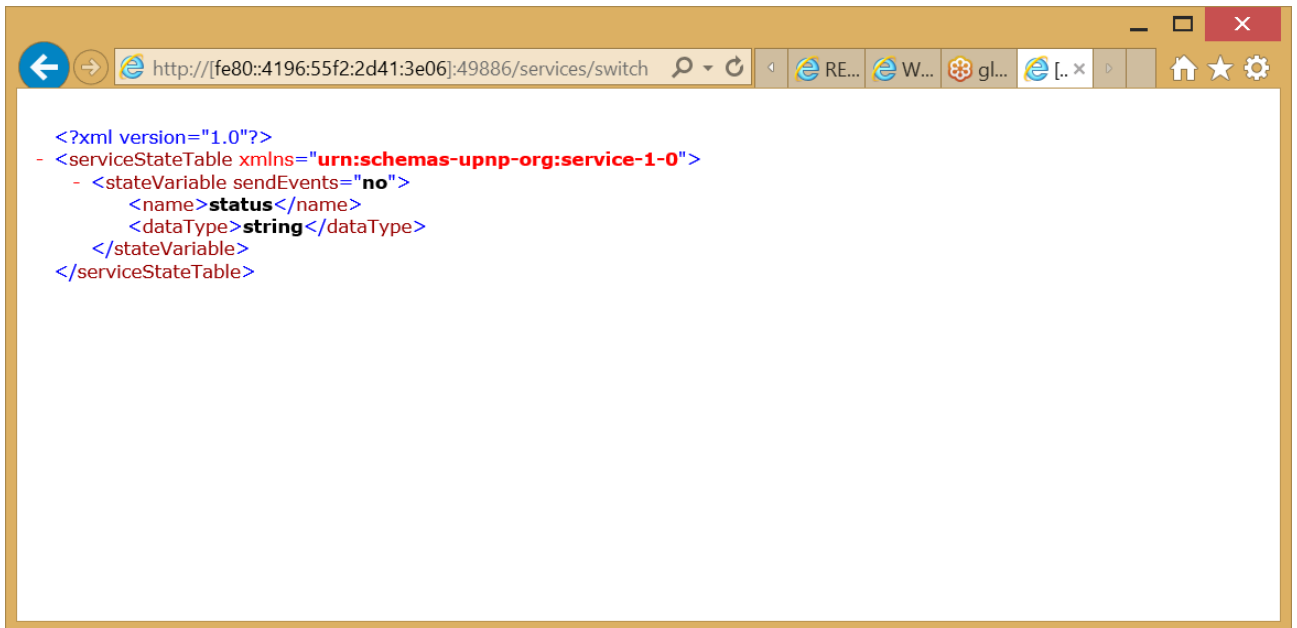


Figure 2: Statevariables listing

### Checking Statevariable Values

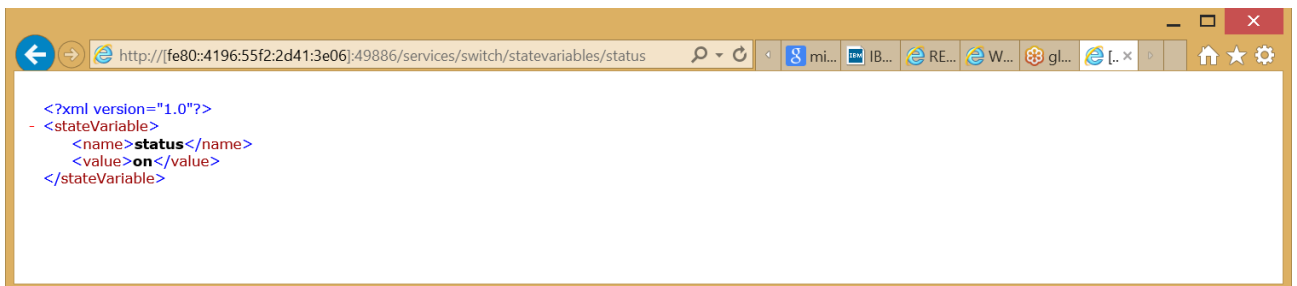


Figure 3: Statevariable value

### Retrieving Actions



```

<?xml version="1.0"?>
- <actionList xmlns="urn:schemas-upnp-org:service-1-0">
  - <action IoTannotation="">
    <name>GetSwitchStatus</name>
    - <argumentList>
      - <argument>
        <name>status</name>
        <direction>out</direction>
        <retval/>
        <relatedStateVariable>status</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  - <action IoTannotation="">
    <name>SetSwitchStatus</name>
    - <argumentList>
      - <argument>
        <name>status</name>
        <direction>in</direction>
        <relatedStateVariable>status</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  - <action IoTannotation="">
    <name>Toggle</name>
  </action>
  - <action IoTannotation="">
    <name>TurnOff</name>
  </action>
  - <action IoTannotation="">
    <name>TurnOn</name>
  </action>
</actionList>
    
```

Figure 4: Action Listing

### Invoking Actions

Actions can be invoked using parameters following the syntax ***action?param1=X&param2=Y***

### Summary

http://<endpoint>/services	Retrieves all services offered by the IoTResource
http://<endpoint>/services/<name>	Retrieves all state variables associated with the service
http://<endpoint>/services/<name>/statevariables	Retrieves the current value of all state variables
http://<endpoint>/services/<name>/statevariables/<name>	Retrieves the value of a specified state variable
http://<endpoint>/services/<name>/actions	Retrieves all actions that can be performed on the IoTResource
http://<endpoint>/services/<name>/actions/<name>?param1=X&param2=Y	Invokes an action with or without parameters.

## 4 IoTResource Catalogue

The IoT Resource Catalogue provides the means to store more elaborate metadata regarding the services compared to the Network Manager Service Catalogue. The IoT Resource Catalogue uses service descriptions that are expressed in an extended version of SCPD (Service Control Protocol Description) which is the standard for service descriptions in DLNA/UPnP. An example of the SCPD description is shown below, see Listing 1.

The reason for using the extended SCPD format is that it is well defined and used for service discovery as well that it is possible to describe services independently of their implementation. This makes it possible to describe REST based services which do not really have any formal description language.

There are two ways to register an IoTResource, i.e. service, with the IoT Resource Catalogue:

- UPnP Discovery using SSDP and SCPD
- SELF Registration

If an IoTResource supports the UPnP Protocol the IoTResource will register automatically with the IoT Resource Catalogue. If a service is integrated using the developer tools this information will be created mostly automatically and the service will be discovered dynamically by UPnP as well. But it also possible to manually register the service in the IoT Resource Catalogue if one prefers by using the RegisterResource action of the catalogue service of the IoTResourceCatalogue.

```

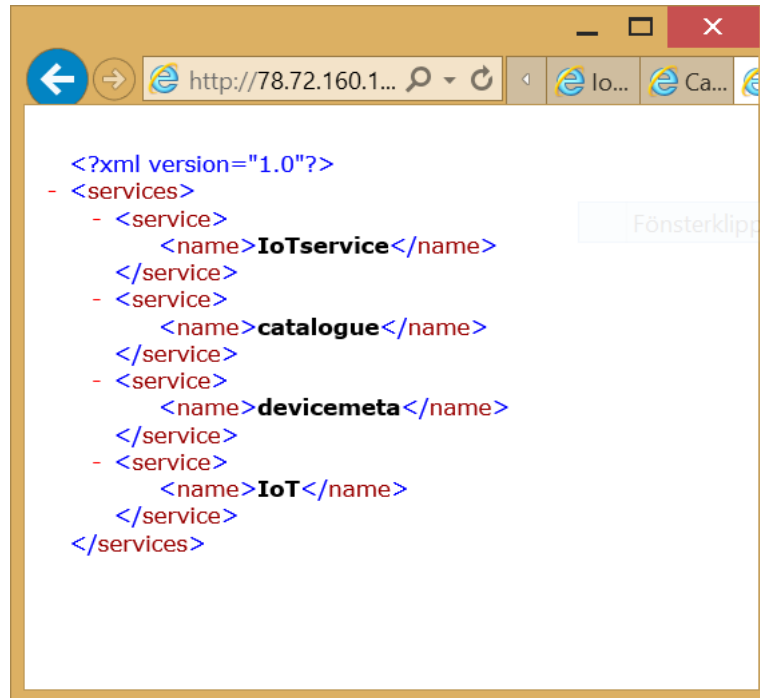
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:IoTdevice:SharedDataSpace:1</deviceType>
    <gateway xmlns="IoT">AIRBUS</gateway>
    <status xmlns="IoT">web service initiated</status>
    <wsendpoint
xmlns="IoT">http://192.168.9.96:8081/S2D2SDevice/S2D2SService</wsendpoint>
    <virtualAddress xmlns="IoT">128.5151.99292.22222</virtualAddress>
    <networkmanager xmlns="IoT" />
    <friendlyName>S2D2SDevice</friendlyName>
    <manufacturer>BRIDGE Integration Meeting</manufacturer>
    <manufacturerURL>http://wwcnet.se</manufacturerURL>
    <modelDescription>Proxy for S2D2s</modelDescription>
    <modelName>S2D2s</modelName>
    <modelNameNumber>1</modelNameNumber>
    <UDN>uuid:caae981e-cf1f-4cf5-bcc7-6849b45144b2</UDN>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:shareddataspace:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:shareddataspace</serviceId>
        <scpd xmlns="urn:schemas-upnp-org:service-1-0">
          <specVersion>
            <major>1</major>
            <minor>0</minor>
          </specVersion>
          <actionList>
            <action IoTannotation="">
              <name>ListSubscriptions</name>
              <argumentList>
                <argument>
                  <name>subscriptions</name>
                  <direction>out</direction>
                  <retval />
                  <relatedStateVariable>Subscriptions</relatedStateVariable>
                </argument>
              </argumentList>
            </action>
          </actionList>
          <serviceStateTable>
            <stateVariable sendEvents="no">
              <name>Subscriptions</name>
              <dataType>string</dataType>
            </stateVariable>
          </serviceStateTable>
        </scpd>
      </service>
    </serviceList>
  </device>
</root>

```

Listing 1: Example of a service description in SCPD

#### 4.1 Retrieving catalogue data

The IoTResource Catalogue offers a number of services which can be listed using the following REST-expression: **http://<catalogueendpoint>/services**. If you type this into a browser the result will be:

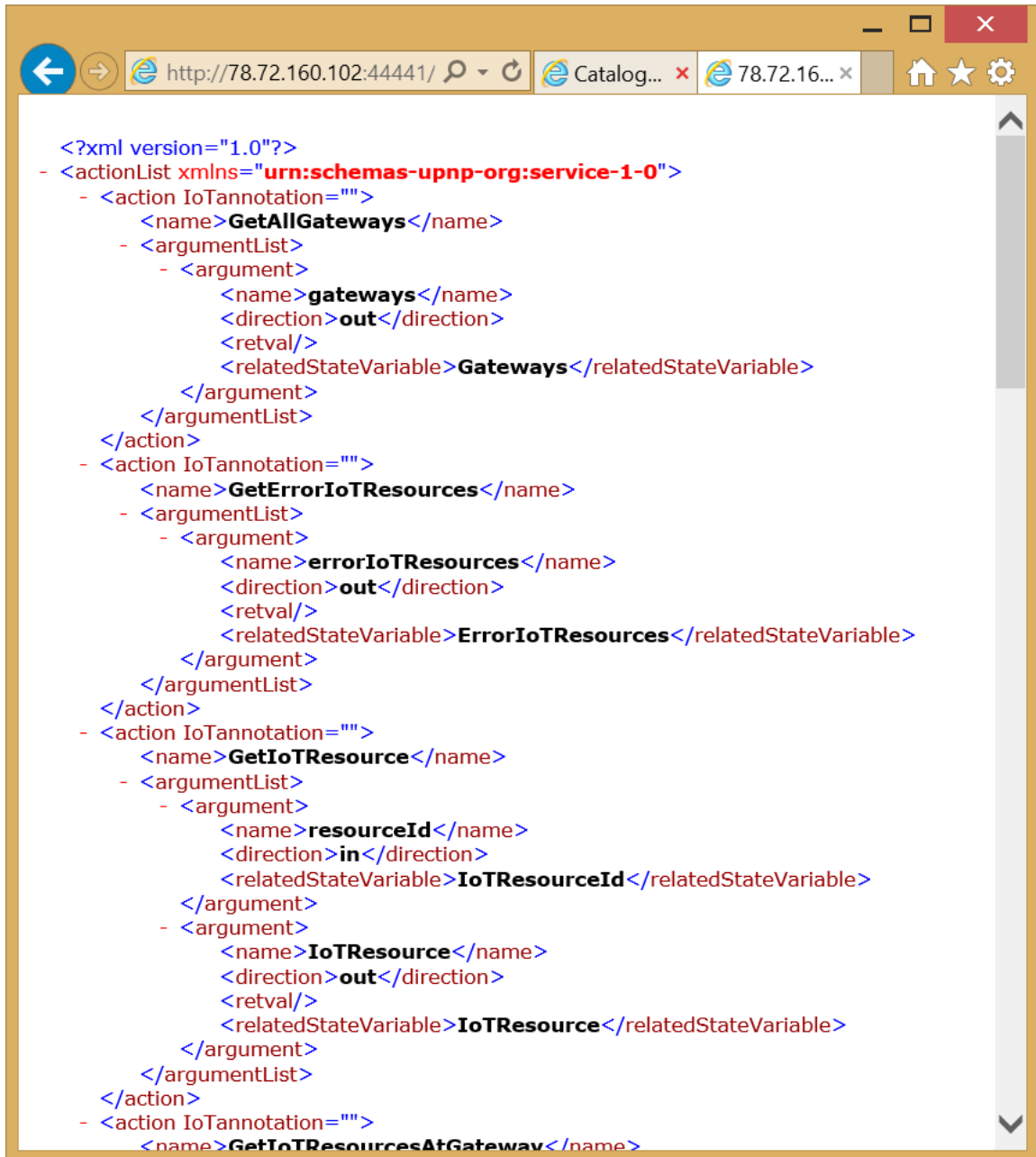
A screenshot of a web browser window showing XML data. The address bar contains the URL 'http://78.72.160.1...'. The main content area displays the following XML structure:

```
<?xml version="1.0"?>
- <services>
  - <service>
    <name>IoTservice</name>
  </service>
  - <service>
    <name>catalogue</name>
  </service>
  - <service>
    <name>devicemeta</name>
  </service>
  - <service>
    <name>IoT</name>
  </service>
</services>
```

Figure 5: IoTResource Catalogue services

Each service provides a number of actions that can be performed on the IoTResource. The catalogue service provides the main functionality of the IoTResource Catalogue. You can list all actions provided by a service with the following REST-expression:

**http://<catalogueendpoint>/services/actions**. The returned XML specifies the action and the arguments needed to call it:



```

<?xml version="1.0"?>
- <actionList xmlns="urn:schemas-upnp-org:service-1-0">
  - <action IoTAnnotation="">
    <name>GetAllGateways</name>
    - <argumentList>
      - <argument>
        <name>gateways</name>
        <direction>out</direction>
        <retval/>
        <relatedStateVariable>Gateways</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  - <action IoTAnnotation="">
    <name>GetErrorIoTResources</name>
    - <argumentList>
      - <argument>
        <name>errorIoTResources</name>
        <direction>out</direction>
        <retval/>
        <relatedStateVariable>ErrorIoTResources</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  - <action IoTAnnotation="">
    <name>GetIoTResource</name>
    - <argumentList>
      - <argument>
        <name>resourceId</name>
        <direction>in</direction>
        <relatedStateVariable>IoTResourceId</relatedStateVariable>
      </argument>
      - <argument>
        <name>IoTResource</name>
        <direction>out</direction>
        <retval/>
        <relatedStateVariable>IoTResource</relatedStateVariable>
      </argument>
    </argumentList>
  </action>
  - <action IoTAnnotation="">
    <name>GetIoTResourcesAtGateway</name>
  
```

Figure 6: IoTResource Catalogue actions

Below is a short explanation of all actions:

### **GetAllGateways**

Returns all gateways known by the catalogue

### **GetErrorIoTResources**

Returns all IoTResources that are in an error state, for instance that have disappeared from the network without telling about it

### **GetIoTResource**

Argument: resourceId

Returns the SCPD for a specified IoT Resource

**GetIoTResourcesAtGateway**

Argument: gateway ID

Returns the SCPD file for all IoT Resources at a specified gateway

**GetIoTResourcesEndpoints**

Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue

**GetIoTResourcesEndpointsFromXpath**

Argument: Xpath expression

Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue that matches the xpath description

**GetIoTResourcesFromXpath**

Argument: Xpath expression

Returns the SCPD file for all IoTResources known by the catalogue that matches the xpath description

**RegisterResource**

Register an IoTResource directly not using UPnPDiscovery.

**GetManualIoTResources**

Returns all IoTResources that has registered themselves and not through UPnP

**GetNumberOfIoTResources**

Returns the number of IoTResources, UPnPDevices, ErrorResources

**RemoveErrorIoTResources**

Instructs the catalogue to release and forget about the IoTResources that are currently in the error list

**ReScan**

Instructs the catalogue to issue a new M-SEARCH command to find new IoTResources in the network

**ReStartCatalogue**

Instructs the catalogue to forget about all IoTResources and ErrorResources and issue a ReScan command

## 4.2 IoTResource Query Language

The IoTResource Catalogue provides a query language for finding IoTResources and their services. This query language is based on the XPath language<sup>1</sup> for querying XML documents, the IoTResource Description files which are based on the SCPD (Service Control Protocol Description) from the UPnP-standard. The IoTResource Catalogue takes an XPath expression and applies it to the SCPD document of the IoTResources. The IoTResources that matches the XPath expression are then returned.

The following namespaces can be used in the xpath expressions:

- upnp
- IoT
- IoTObservation

---

<sup>1</sup> <http://www.w3schools.com/xpath/>

The XPath querying can be used directly in the REST URL:

*http://<catalogueendpoint>/<xpathexpression*

For example,

*http://<catalogueendpoint>/UPnP:serviceType [.= 'urn:schemas-upnp-org:service:shareddataspace:1']*

returns all IoTResources that of the catalogue at that support the service "urn:schemas-upnp-org:service:shareddataspace:1"

<a href="http://192.168.9.15:44441/*">http://192.168.9.15:44441/*</a>	List all available resources
<a href="http://192.168.9.15:44441//UPnP:serviceType [.= 'urn:schemas-upnp-org:service:shareddataspace:1']">http://192.168.9.15:44441//UPnP:serviceType [.= 'urn:schemas-upnp-org:service:shareddataspace:1']</a>	List all shared dataspace services known to the catalogue
<a href="http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet']">http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet']</a>	List all resources from manufacturer CNet
<a href="http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet'][UPnP:serviceType [.= 'urn:schemas-upnp-org:service:shareddataspace:1']">http://192.168.9.15:44441//upnp:device[upnp:manufacturer='CNet'][UPnP:serviceType [.= 'urn:schemas-upnp-org:service:shareddataspace:1']</a>	List all resources from manufacturer CNet running at a gateway ARMSTRONG and that is currently consuming more than 100 W.

### 4.3 Actuating on IoTResources using the service catalogue

The IoTResource Catalogue provides two means to actuate on an IoTResource:

- By providing the unique identifier of a specific IoTResource
- By providing the local endpoint of a specific IoTResource
- By providing a semantic description that matches one or several IoTResource

#### Using identifier

The keyword is IoTResource to be supplied directly after the <catalogueendpoint> followed by the specific identifier and then the actuation expression:

***http://<catalogueendpoint>/IoTResources/<IoTResourceid>/services/switch/actions/TurnOn***

#### Using endpoint

By first calling the action GetIoTResourcesEndpoint in the catalogue service of the IoTResourceCatalogue you will be able to retrieve the local endpoint of the device and can start communicating directly with the IoTResource. This method is recommended if you need to do a massive amount of calls to IoTResource and then you don't have to go through the central IoTResourceCatalogue. However, it will only work if your Application is executing in the same local network as the IoTResources.

### Using Semantic Description

Provide an xpath expression directly after the <catalogueendpoint> followed by the actuation expression:

***http://<catalogueendpoint>/<xpath>/services/switch/actions/TurnOn***

### Example

Turn on all IoTResources that are of type switchdevice:

***http://<catalogueendpoint>//upnp:devicetype[contains(.,'switchdevice')]/services/switch/actions/TurnOn***

### Example

Turn off all IoTResources that are from CNet and that currently consumes more than 200W:

***http://<catalogueendpoint>//upnp:device[upnp:manufacturer='CNet']][IoT:currentconsumption>200]/services/switch/actions/TurnOff***

### Notes

One current limitation when using semantic descriptions is that the xpath expression cannot contain the word services.

## 4.4 Enabling a service in the IMPRESS platform

There are three main ways of enabling a service in the IMPRESS platform, i.e. making a service available and accessible on the IMPRESS network:

- Creating a proxy that communicates with service using the IoTResource Builder.
- Incorporating the code generated by the IoTResource Builder into the service.
- Enabling the service using the available IMPRESS Middleware services.

The two first options involving the IoTResource Builder are the easiest way to enable the service in the IMPRESS network. The code stubs generated already contain the code for registering the service in the Service Catalogue and it has automatically produced the service description that will be part of the IoTResource Catalogue. The usage of the tools is described in section 5.1.

Therefore this section will deal with what is necessary for the third option where the service is enabled by using the IMPRESS Middleware services. The integration with the IMPRESS system and network can be done at two different depths depending on ambition level and need. In the simple integration the service is discoverable and can be invoked using the IMPRESS network but the caller must now what the service and its API beforehand. In the full integration the service provides additional metadata describing the service making it possible to dynamically determine the purpose and API of the service.

The two following sub section will detail how a service can be IMPRESS network enabled by using the IMPRESS Middleware services.

### 4.4.1 Enabling a service on the IMPRESS network

In order to register a service on the IMPRESS network the service endpoints must be known. The endpoint is the URL where the service can be accessed by a client application. The service can have multiple endpoints, for example in order to make it available using different protocols, for instance one for REST and one for Web Service protocols. Each of the services endpoints needs to be registered individually in the Network Manager Service Catalogue. The process is repeated for each endpoint according to these steps:



- Create the meta description for the endpoint. This is a list of key value pairs with properties. Two values are compulsory in all registrations: DESCRIPTION that describes the endpoint, for instance S2D2sCNet:REST; SID The service identity, for instance urn:http:ws:IMPRESS:Middleware:SharedDataSpace:1, this attribute describes which interface/service is implemented.
- Make a registration call to the local IMPRESS Network Manager to register the service endpoint. Note that the supplied endpoint must be accessible for the Network Manager to reach, otherwise the service can never be called.

```

//Connect to the Service Catalogue
ServiceCatalogue.NetworkManager sc = new ServiceCatalogue.NetworkManager();
//Using the local network manager
sc.Url = "http://localhost:9090/cxf/services/NetworkManager";

//Using the local network manager
ServiceCatalogue.Part[] parts = new ServiceCatalogue.Part[5];
ServiceCatalogue.Part p = new ServiceCatalogue.Part();
//Create the DESCRIPTION (Mandatory key)
p.key = "DESCRIPTION";
p.value = "S2D2SDevice:StaticWS";
parts[0] = p;

//Create the SID (Mandatory key), Service ID
p = new ServiceCatalogue.Part();
p.key = "SID";
p.value = "urn:http:ws:IMPRESS:Middleware:SharedDataSpace:1";
parts[1] = p;

//Create the PID (Optional key), Persistent ID. Needs to be a unique name on
the IMPRESS network.
p = new ServiceCatalogue.Part();
p.key = "PID";
p.value = "my unique id";
parts[2] = p;

//Examples of additional keys
p = new ServiceCatalogue.Part();
p.key = "HOST_NAME";
p.value = Environment.MachineName;
parts[3] = p;

p = new ServiceCatalogue.Part();
p.key = "START_TIME";
p.value = DateTime.Now.ToString(); ;
parts[4] = p;

//Make the registration
ServiceCatalogue.Registration rid = sc.registerService(parts, m_wsendpoint,
"eu.linksmart.network.grand.impl.GrandMessageHandlerImpl");

HID = rid.virtualAddressAsString;
System.Console.WriteLine("Virtual Address:" + HID);

```

Listing 2: Example of registering a service endpoint in C#

Listing 2 shows the steps in code where `m_wsendpoint` contains the endpoint for the service. The SID (service identity) can be common for many services in the IMPRESS network when they implement the same service interface (API). When this registration is done the service and its endpoint is available to the whole IMPRESS network.

### 4.4.2 Registering Service Metadata in the IoTResource Catalogue

Using the IoTResource Builder services automatically register to the IoT Resource Catalogue by using the standard UPnP discovery mechanism. This section will describe how to register the Service Metadata using IMPRESS middleware service invocations in code.

The actual call to register the services is simple but some extra functionality must be implemented in order for the service to be properly registered. This requires a small understanding on how the SCPD works in relation with UPnP and how the IoTResource Catalogue deals with service descriptions.

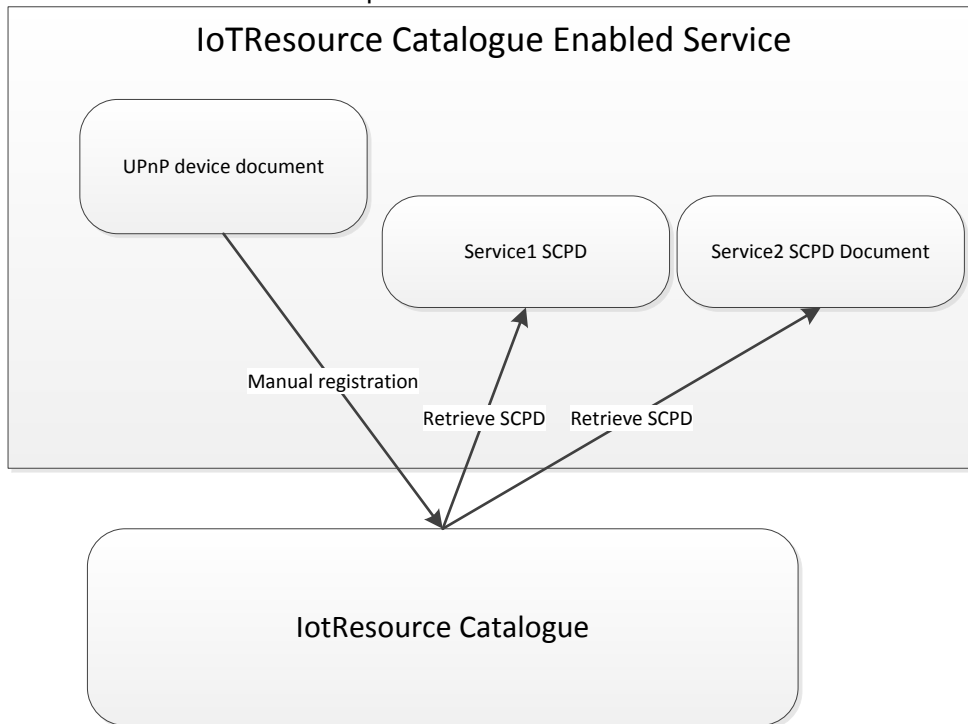


Figure 7: Manual IoTResource Catalogue registration

Figure 7 shows how the IoTResource Catalogue finds services manually, the steps are as follows:

- First the service makes a manual registration of the services it provides, this registration contain links to SCPD files describing each of the individual services.
- The IoTResource Catalogue retrieves the description of the services and adds them to the catalogue.

This means that the service must be able to publish the SCPD files for HTTP based retrieval. This can be done by using any HTTP-based Web Server that can be accessed from the IoTResource Catalogue.

The UPnP Device document that is provided for the manual registration is a standard UPnP device document.

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:IoTresource:testservice:1</deviceType>
    <friendlyName>TestService</friendlyName>
    <manufacturer>IMPRESS</manufacturer>
    <modelName>Test</modelName>
    <modelName>1</modelName>
    <UDN>uuid:15696574-1a4d-42e0-8907-bee3e110e2f1</UDN>
    <serviceList>
```

```

    <service>
      <serviceType>urn:schemas-upnp-org:service:testservice:1</serviceType>
      <serviceId>urn:serviceId:testservice:1</serviceId>
      <SCPDURL>http://127:0.0.1:7237/serviceId-testservice-1_scpd.xml</SCPDURL>
    </service>
  </serviceList>
</device>
</root>

```

Listing 3: Example of XML document used to register service

Listing 3 shows a simple example device document that is used to register a service. In this example only one service is provided "urn:serviceId:testservice:1", but more services can be added to the serviceList. The most important parts that need to be entered correctly in the document are:

- friendlyName: This is the name that service will have in different service browser
- serviceId: Should have the same content as the SID used to register in the Network Manager Service Catalogue
- SCPDURL: This is the link to the SCPD document describing the service, this URL must be accessible for the IoTResource Catalogue.

The SCPD document for the service needs to be manually created using an XML editor, the actual syntax and also links to some tools can be found at the UPnP forum web site <http://www.upnp.org/>. A very simple example of an SCPD document is shown in Listing 4.

```

<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>TestMethod</name>
      <argumentList>
        <argument>
          <name>testInput</name>
          <direction>in</direction>
          <relatedStateVariable>Test</relatedStateVariable>
        </argument>
        <argument>
          <name>testResponse</name>
          <direction>out</direction>
          <retval />
          <relatedStateVariable>Result</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="no">
      <name>Test</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>Result</name>
      <dataType>boolean</dataType>
    </stateVariable>
  </serviceStateTable>
</scpd>

```

Listing 4: Example SCPD

The example SCPD can be extended with the service annotations described in section 5.1.1 to add additional metadata to the service.

Finally we show the code necessary to register the service in the IoTResource Catalogue using the IMPRESS Middleware API.

```
public void RegisterService()
{
    XmlDocument xDeviceDocument = new XmlDocument();
    xDeviceDocument.Load("mydeviceDocument.xml");

    IoTResourceCatalogue.ApplicationDeviceManager
        IoTResCat = new IoTResourceCatalogue.ApplicationDeviceManager();

    IoTResCat.AddDevice(xDeviceDocument.InnerXml /*xml as string*/);
}
```

Listing 5: Manual Service Registration in the IoTResource Catalogue

As shown in Listing 5 the actual call to make the registration is simple but the complexity lies in the creation of the SCPD files and to make sure that the description matches the actual implementation. Therefore we recommend the usage of the IoTResource Builder tool when creating or enabling services on the IMPRESS network because the SCPD and registration information will be created by the tool.

## 5 Development Tools

### 5.1 IoTResource Builder and service annotations

The IMPRESS middleware provides access to the set of ICT resources in an energy system context, such as different sensing devices, data repositories, social media streams, UAVs etc. The middleware service layer provides client applications uniform access to all such resources (below referred to as IoTResources, which is the LinkSmart resource concept). The IoTResource Builder allows a developer to define IoTResources and automatically generate the necessary IoTResource code stubs. Services can then be built using these IoTResources. The IoTResources will also automatically register themselves in both the Network Manager Service Catalogue as well as the IoTResource Catalogue.

Complete description, tutorials and download of the IoTResource Builder are available at:

<http://www.iotworldservices.com/wiki/iotworldserviceswiki/iot-resource-builder/iotresource-builder/>

#### 5.1.1 Service annotations

In order to facilitate the use of IMPRESS services both in run-time and in design time, the platform supports the annotation of both services and resources. Annotations in this context means the possibility to associate various semantic descriptions to IoTResources via their service access points.

The annotations can be made searchable for developers as an aid in service development.

They can also be used to facilitate the resource discovery processes, and service matching for potential application clients. The annotations are included in the service definitions, which are used as input to the code generation process, which creates program stubs for IoTResources

There are different levels of service annotations.

- Service Summary, a description of the overall function of a service, including references to standards or other external sources.
- Service Actions. Each service has one or more actions (operation /methods). Each action implements some sensing or actuation function. Annotations include action purpose, and arguments and results.
- Property Level (state variables). The arguments and results, the state variables, can also be described in more detail, including their value sets and references to standards.
- Effect annotations. Actions can also be annotated with a list of possible effects they might have in the applications context, or more specifically on other state variables. As an example, turning off a fan might cause a temperature raise, and perhaps also a decrease in energy consumption.

There are numerous approaches to service description frameworks. The service description tool does not impose the use of any specific service annotation standard, but rather encourages the referencing to domain specific standards, controlled vocabularies (or ontologies), in the annotations of the service semantics, e.g., emergency messaging data set standards like EDXL.

From a structural and syntactical view, the service description is based on the UPnP<sup>2</sup> device descriptions and the SCPD format, and USDL<sup>3</sup>.

#### 5.1.2 Use of The resource builder

The IoTResource Builder allows you to define your IoTResources and automatically generate the necessary IoTResource code stubs. Services can then be built using these IoTResources.

---

<sup>2</sup> <http://www.upnp.org/>

<sup>3</sup> Unified Service Description Language, <http://linked-usdl.org/>

The following sections give examples of how a service can be described using the Resource Builder tool.

**Service Summary**

This service will be mapped to an IoTResource which monitors in-door air quality, using a CO<sub>2</sub> Sensing device in conference room. We start by providing the overall description, the Service Summary.

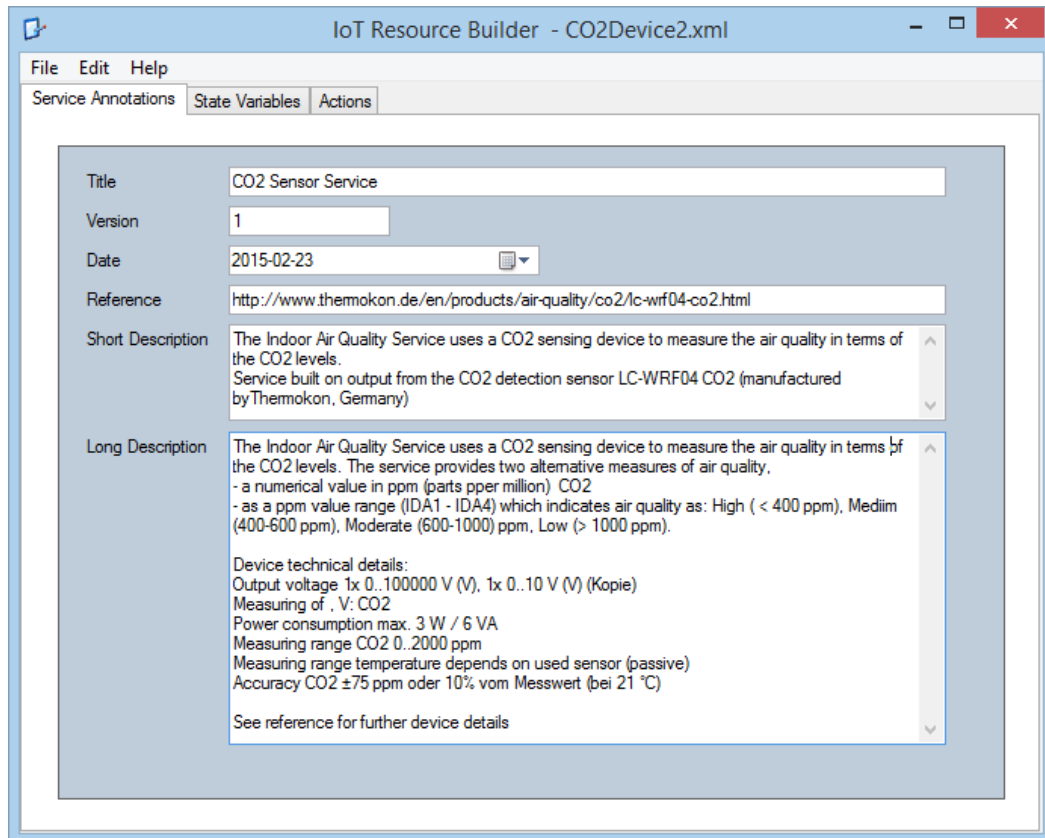


Figure 8: Service Summary description using the IoTResource Builder

The annotations are encoded in an XML vocabulary which will be associated to the IoTResource in the code generation process (see below).

The Service Summary description is shown in its corresponding XML encoding below.

```

<serviceAnnotations xmlns="IoT">
  <name>Indoor Air Quality Service</name>
  <version>1</version>
  <date>2015-02-23</date>
  <shortDescription>
    The Indoor Air Quality Service uses a CO2 sensing device to measure the air quality
    in terms of the CO2 levels. Service built on output from the CO2 detection sensor LC-
    WRF04 CO2 (manufactured byThermokon, Germany)
  </shortDescription>
  <longDescription>
    The Indoor Air Quality Service uses a CO2 sensing device to measure the air quality in
    terms of the CO2 levels. The service provides two alternative measures of air quality,
    - a numerical value in ppm (parts pper million) CO2
    - as a ppm value range (IDA1 - IDA4) which indicates air quality as: High ( &lt; 400
    ppm), Mediim (400-600 ppm), Moderate (600-1000) ppm, Low (&gt; 1000 ppm).

    Device technical details:
    Output voltage 1x 0..100000 V (V), 1x 0..10 V (V) (Kopie)
    Measuring of , V: CO2
    Power consumption max. 3 W / 6 VA
    Measuring range CO2 0..2000 ppm
  </longDescription>
</serviceAnnotations>

```

```

Measuring range temperature depends on used sensor (passive)
Accuracy CO2 ±75 ppm oder 10% vom Messwert (bei 21 °C)
See reference for further device details
</longDescription>
<referenceUrl>
http://www.thermokon.de/en/products/air-quality/co2/lc-wrf04-co2.html
</referenceUrl>
</serviceAnnotations>
    
```

Figure 9: Service Summary XML

**Actions**

Each action (similar to operations/methods) of a service may also have their own annotations specified. This example shows an action for reporting the air quality in ppm (parts per million) CO<sub>2</sub> based on a standard for indoor air quality (IDA<sup>4</sup>).

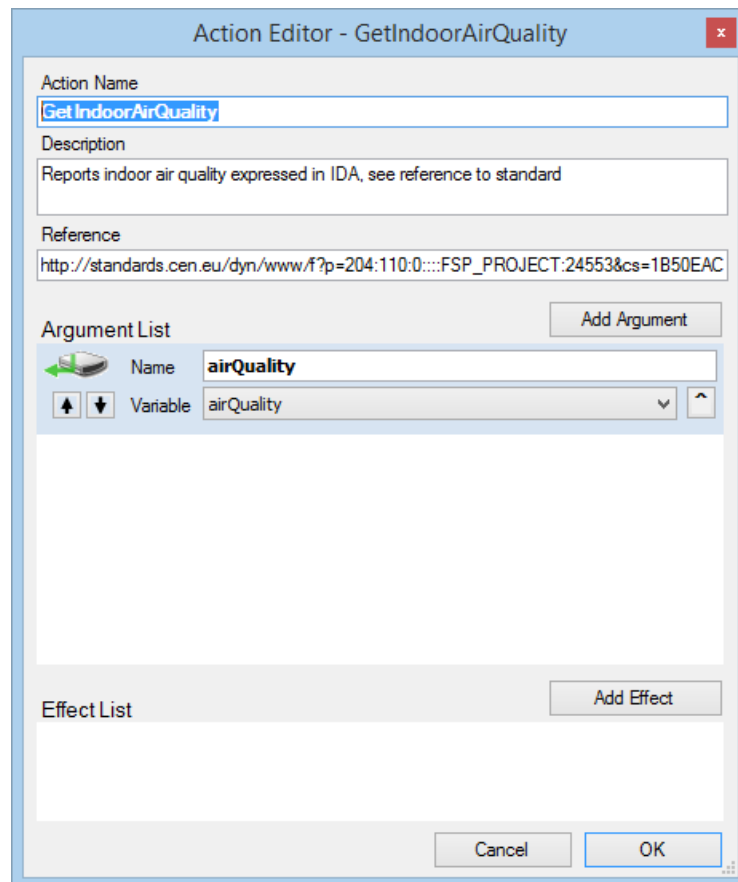


Figure 10: Action annotation

The corresponding XML follows. It also contains two additional actions for the service.

```

<actionmetadata xmlns="IoT">
  <actionList>
    <action name="GetIndoorAirQuality">
      <description>
        Reports indoor air quality expressed in IDA, see reference to standard.
      </description>
      <referenceUrl>
http://standards.cen.eu/dyn/www/f?p=204:110:0:::FSP\_PROJECT:24553&cs=1B50EAC84642115F35A7D9F005762E46B
      </referenceUrl>
      <effects>
    
```

<sup>4</sup> <http://www.aafeurope.com/en/155/en13779-standard>

```

        No effects reported
    </effects>
</action>
<action name="GetCO2Level">
    <description>
        Reports indoor air quality expressed as level of CO2 in ppm
    </description>
    <referenceUrl></referenceUrl>
    <effects>
        <effect>
            <stateVariable></stateVariable>
            <description>text explaining possible effect</description>
            <referenceUrl></referenceUrl>
        </effect>
    </effects>
</action>
<action name="TurnOffCO2Sensor">
    <description>
        Turns off the CO2 Sensor Device and returns the current CO2Level
    </description>
    <referenceUrl></referenceUrl>
    <effects>
        <effect>
            <stateVariable></stateVariable>
            <description>
                Device is turned off. Last measurement accessible in log.
            </description>
            <referenceUrl></referenceUrl>
        </effect>
    </effects>
</action>
</actionList>
</actionmetadata>

```

Figure 11: Action annotation XML

As mentioned above, it also possible to describe any additional effects an action might have. Note that these “effects” are not to be seen as hard dependencies between actions/ state variables maintained by the service run-time, but rather as a way to document possible effects in the application context.

In the example above (TurnOffCO2Sensor), the “effect” simply states that at (power) turn off, the last measured value is available as an IoTObservation from logged data.

### Properties (State Variables)

The inputs/outputs of a service are represented by state variables associated with each of the actions. The Air quality action above reports measurements to be interpreted according to a standard for in-door air quality using intervals of ppm ranges.



Figure 12: State variable annotation

The corresponding XML encoding for this State variable annotation is shown below.

```

<statevariablemetadata xmlns="IoT">
  <statevariablelist>
    <statevariable name="airQuality">
      <IoTEvent>true</IoTEvent>
      <!--checkbox Event: Generates IoT event when state changes -->
      <IoTStored>true</IoTStored>
      <!-- checkbox Logged:Store state changes automatically using storage manager -->
      <IoTUoM>IDA</IoTUoM>
      <!-- Valueset? - Unit of Measurement C, cm, kg....etc -->
      <description>
        Indoor air (IDA) quality in PPM intervals according to the EN13779 standard
      </description>
      <referenceUrl>
        http://standards.cen.eu/dyn/www/f?p=204:110:0:::FSP_PROJECT:24553&cs=1B50EAC84642115F35A7D9F005762E46B
      </referenceUrl>
      <valueset>
        <entry>
          <value>IDA4</value>
          <description>(Low) CO2Level more than 1000 PPM</description>
        </entry>
        <entry>
          <value>IDA3</value>
          <description>(Moderate) CO2Level btw 600-1000</description>
        </entry>
        <entry>

```

```

        <value>IDA2</value>
        <description>(Medium) CO2Level btw 400-600 PPM</description>
    </entry>
    <entry>
        <value>IDA1</value>
        <description>(High) CO2Level less than 400 PPM</description>
    </entry>
</valueset>
<vsReferenceUrl></vsReferenceUrl>
</statevariable>
Additional variables here...
</statevariableList>
</statevariablemetadata>
    
```

Figure 13: Annotations for State Variables

### 5.1.3 Components in the IoTResource Builder

The IoTResource Builder is built on two separate components, see Figure 14:

- The IoTResource Builder GUI
- The IoT Code Generator

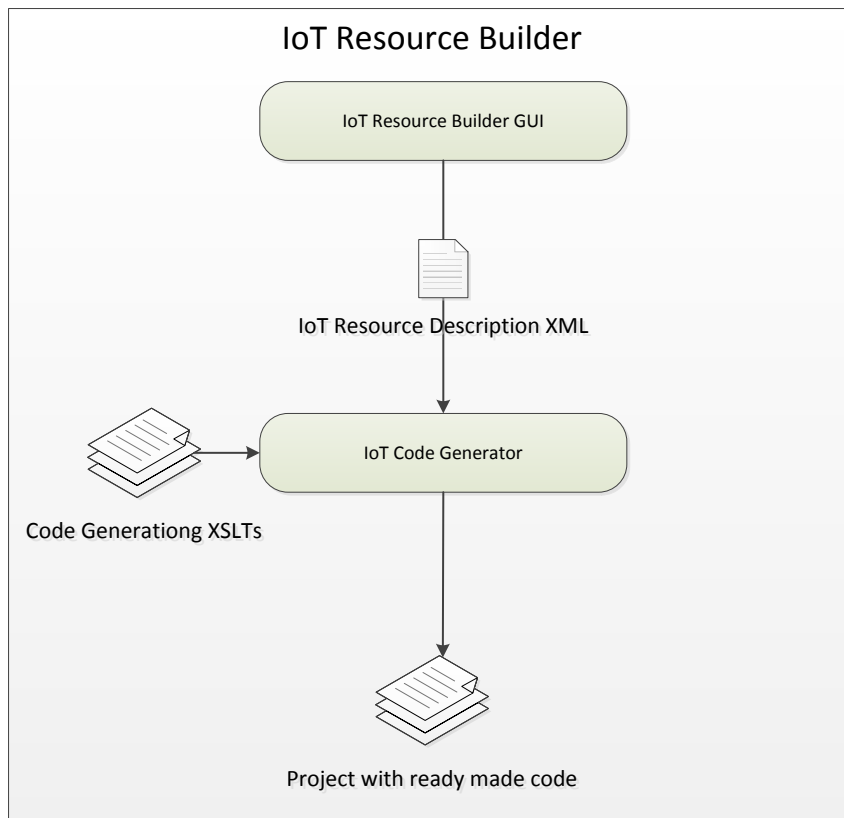


Figure 14: IoTResource Builder Components

The IoTResource Builder GUI creates an IoTResource Description XML which is then sent to the IoT Code Generator to create the code. The reason for this division is to make the IoT Code Generator reusable for other tools, for instance the GUI could be replaced by a completely web based interface but still using the same code generation.

The IoTResource Description XML is based on the UPnP device XML but with some small differences. Firstly the IoTResources service description (SCPD) is in lined in the Device XML. Secondly there is an envelope which carries some code generation meta data, see Figure 15.

```

<device>
  <deviceType>urn:schemas-upnp-org:IoTresource:CO2Sensor:1</deviceType>
  <friendlyName>CO2Sensor</friendlyName>
    
```

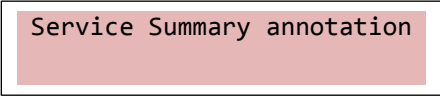
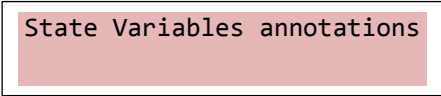
```

<manufacturer>LinkSmart Open Source</manufacturer>
<manufacturerURL>
  http://www.iotworldservices.com/wiki/iotworldserviceswiki/
</manufacturerURL>
<modelDescription>
  CO2Sensor UPnP Device Using Auto-Generated UPnP Stack
</modelDescription>
<modelName>CO2Sensor Device</modelName>
<modelNumber>X1</modelNumber>
<productCode>CO2Sensor-X1</productCode>
<serviceList>
  <service>
    <serviceName>CO2SensorProject</serviceName>
    <serviceType>urn:schemas-upnp-org:service:CO2Sensor::1</serviceType>
    <serviceId>urn:upnp-org:serviceId:CO2Sensor</serviceId>
    <SCPD>
      <specVersion xmlns="urn:schemas-upnp-org:service-1-0">
        <major>1</major>
        <minor>0</minor>
      </specVersion>
      <actionList xmlns="urn:schemas-upnp-org:service-1-0">
        <action>
          <name>GetCO2Level</name>
          <argumentList>
            <argument>
              <name>CO2Level</name>
              <direction>out</direction>
              <retval />
              <relatedStateVariable>CO2Level</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
        <action>
          <name>GetIndoorAirQuality</name>
          <argumentList>
            <argument>
              <name>airQuality</name>
              <direction>out</direction>
              <retval />
              <relatedStateVariable>airQuality</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
        <action>
          <name>TurnOffCO2Sensor</name>
          <argumentList>
            <argument>
              <name>CO2Level</name>
              <direction>in</direction>
              <relatedStateVariable>CO2Level</relatedStateVariable>
            </argument>
          </argumentList>
        </action>
      </actionList>
      <serviceStateTable xmlns="urn:schemas-upnp-org:service-1-0">
        <stateVariable sendEvents="no">
          <name>_IoTActionMetaData_</name>
          <dataType>string</dataType>
          <defaultValue>
            <?xml version="1.0" encoding="utf-8"?>
              <actionmetadata xmlns="IoT">
                <actionList>

```

Actions Annotations

```

        </actionList>
      </actionmetadata>
    </defaultValue>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>CO2Level</name>
    <dataType>i2</dataType>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>_IoTResourceMetaData_</name>
    <dataType>string</dataType>
    <defaultValue>
      <serviceAnnotations>

      </serviceAnnotations>
    </defaultValue>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>_IoTStateVariableMetaData_</name>
    <dataType>string</dataType>
    <defaultValue>
      <?xml version="1.0" encoding="utf-8"?>
      <statevariablemetadata xmlns="IoT">
        <statevariableList>

        </statevariableList>
      </statevariableList>
    </defaultValue>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>airQuality</name>
    <dataType>string</dataType>
  </stateVariable>
</serviceStateTable>
</SCPD>
</service>
</serviceList>
</device>
</root>
</upnp>
</DeviceInfo>

```

Figure 15: Example of an IoTResource Description XML. Coloured rectangles represent the annotation sections.

There are four specific tags in the Environment section that controls the code generation:

- *CodeNameSpace*: The namespace used for the code generated, usage depends on target language.
- *ProjectName*: The name used for the resulting code project.
- *ClassName*: Class name stem used for the generated classes for the IoTResource.
- *IoTResourceType*: Decides which type IoTResource code is generated, current possible values are IoTDevice, IoTService and IoTThing.

The actual code generation is performed by using XSLT transformations using the IoTResource Description XML as input. The set of XSLT transformation<sup>5</sup> create the output files that are part of the resulting development project solution.

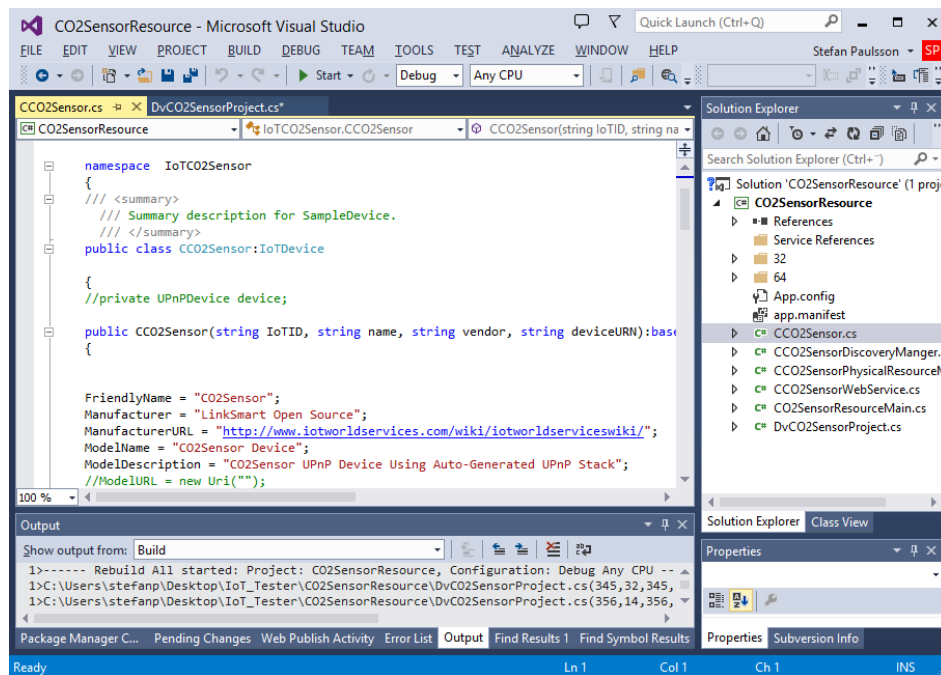


Figure 16: Development environment generated

Initially C# Visual Studio projects are supported as target environment. However, with the use of XSLT this can be easily extended to support other languages such as java, swagger.

### 5.2 IoTResource Catalogue

The IoTResource Catalogue discovers and keeps track of available IoTResources in the network and their service descriptions. It provides a REST and Web Service based interface to select and retrieve data about the IoTResources and their services. As an example see Figure 17 below that shows which IoTResources have been discovered on the gateway "KURSAAL", which handles several physical gateways (KURSAAL, ELO2,CLEMONS) and which IoT Services they offer. IoTResources are discovered and managed by the IoT Resource Catalogue.

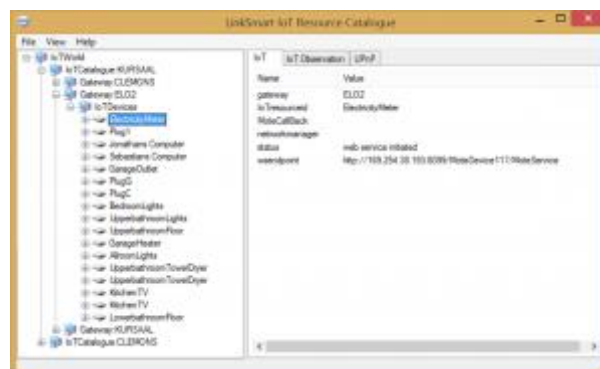


Figure 17: IoT Resource Catalogue

<sup>5</sup> See LinkSmart code repository <https://linksmart.eu/redmine/>



**GetErrorIoTResources**

Returns all IoTResources that are in an error state, for instance that have disappeared from the network without telling about it

**GetIoTResource**

Argument: resourceId

Returns the SCPD for a specified IoT Resource

**GetIoTResourcesAtGateway**

Argument: gateway ID

Returns the SCPD file for all IoT Resources at a specified gateway

**GetIoTResourcesEndpoints**

Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue

**GetIoTResourcesEndpointsFromXpath**

Argument: Xpath expression

Returns the IotResourceId, FriendlyName and the localendpoint for all IoTResources known by the catalogue that matches the xpath description

**GetIoTResourcesFromXpath**

Argument: Xpath expression

Returns the SCPD file for all IoTResources known by the catalogue that matches the xpath description

**RegisterResource**

Register an IoTResource directly not using UPnPDiscovery.

**GetManualIoTResources**

Returns all IoTResources that has registered themselves and not through UPnP

**GetNumberOfIoTResources**

Returns the number of IoTResources, UPnPDevices, ErrorResources

**RemoveErrorIoTResources**

Instructs the catalogue to release and forget about the IoTResources that are currently in the error list

**ReScan**

Instructs the catalogue to issue a new M-SEARCH command to find new IoTResources in the network

**ReStartCatalogue**

Instructs the catalogue to forget about all IoTResources and ErrorResources and issue a ReScan command

### 5.3 IoTResource Catalogue Browser.

IoT Resource Catalogue Browser provides a user interface to look and interact with IoT resources in the network node. Basically it provides a user interface to the IoT Resource Catalogue. The IoT Resource Catalogue Browser can be used for looking at the service descriptions and also to invoke actions in the service (If the service supports this) Complete description examples and downloads of the IoT Resource Catalogue Browsers are available at: <http://www.iotworldservices.com/wiki/iotworldserviceswiki/iotresource-catalogue-browsers/>.

When you double click on the executable it browser will first discover the IoTResource Catalogue in your local network. If you click on the catalogue name in the tree, you will see three tabs to the right. The first tab shows you the number of IoTResources this catalogue has discovered.

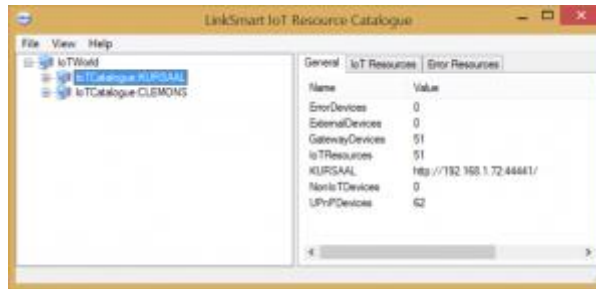


Figure 20: Initial Windows IoT Resource browser window

The second tab shows the IoTResourceIds and the endpoints to the different IoTResources. In case there are IoTResources which are in some error state and therefore cannot be accessed, they will be listed in the third tab.

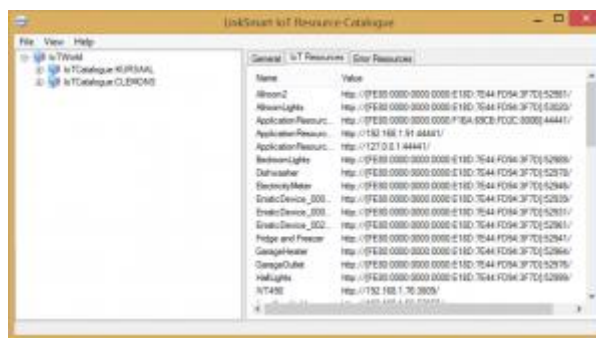


Figure 21: Second Tab with IoT Resources end points

You can now expand the tree on the left. The gateway nodes correspond to different hardware gateways (normally computers) in your network which hosts the IoTResources. If you click on one IoTResource, you will see three tabs to the right. The first tab (IoT) lists the state variables/properties that are specific for LinkSmart.

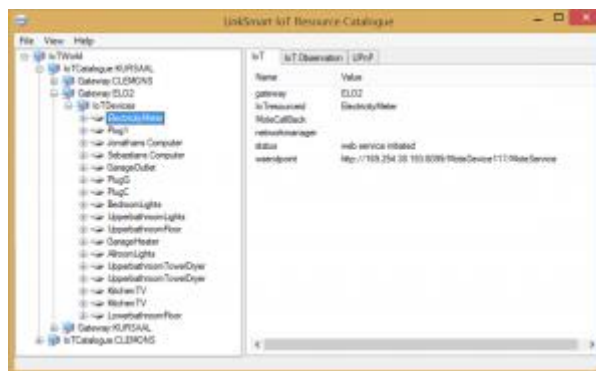


Figure 22: Expanded view



## 6 References

Use the following style for references.

- (EC, 2007) European Commission (2007). A lead market initiative for Europe. Brussels. COM(2007) 860 final.
- (Milagro et al 2008) Milagro, F., Antolin, P., Kool, P., Rosengren, P., Ahlsén M. (2008). SOAP tunnel through a P2P network of physical devices, Internet of Things Workshop, Sophia Antopolis.
- (Chen et al 2007) Chen, Y.C., Liu, C.H., Wang, C.C., Hsieh, M.F. (2007). "RFID and IPv6-enabled Ubiquitous Medication Error and Compliance Monitoring System", 9th International Conference on e-Health Networking, Application and Services, 2007, 19-22 June 2007 Page(s):105 - 108.