ThingToData API Tutorial (under construction)

1 GENERAL INFORMATION AND TERMS

1.1 INVENTORY

Thing2Data Inventory is a repository (database) which stores things. Inventory data is accessed by Inventory API which is REST API.

Inventory is not conventional relational database. There are no tables and no rows. Inventory contains thing identified by unique thingId. There are relations between things, for example thing can 'contain' other thing which is contained by some another thing.

There is no traditional query APi to inventory. You can't make query like 'get all things which title begins with shovel' or 'get first 10 things made by Fiskars ordered by manufacture date descending'. You need to know exact thingId or you can navigate from one thing to another by relations or if you know geolocation you can ask which things are near this location.

1.2 USING INVENTORY - THINK NEWLY AND FORGET ALL YOU KNOW ABOUT DATABASES AND OBJECT-ORIENTED PROGRAMMING!

When planning the features and architecture of your Thing2Data -application, think carefully what Things you will have there. If we talk about gardening applications, Things could be plants, areas in the garden, gardening tools, shops, clubs, service providers etc.

It is important to understand that the user can only access Things for which he has been given access rights to. Thus, there is generally no searching tools. If you wish that your Thing would be visible to others in a way where it is a member of a group that can be browsed, you need to create the group as a Thing and give rights to that (Group)Thing. If you gain access rights to the group and the group has access rights to other Things, you can use those rights.

When you design the logic of your application, please consider Service Requests for all interaction between Things. Thus, when a gardener views her plants, and wishes some help, it should be the plant-Thing's Service Request, which sends the appropriate Service Actions to those service-Things, which are available. And how those Service Requests are created – they are most naturally created when the plant Thing gives Access rights to the service Thing initiates a Service Request with the service Thing, and then the application servicing the service Thing creates the Service Request for the plant Thing.

What all this means actually is that you should plan the Thing-architecture and interactions between the Things using the concepts that are supported by Thing2Data instead of hardwiring the logic to your application. This way you gain the support of Thing2Data for security and logging, and this way the logic supported by Service Requests is available to other applications too. Actually, a wide variety of functionality can be achieved this way via even generic Thing2Data apps if the Thing-architecture and Service Request -configuration is cleverly performed.

Next some terms you should understood to successfully use ThingToData inventory API.

1.3 THING

Thing is individual item which is identified by unique identifier. Thing can be anything, for example suitcase, car, user, shovel, flower and so on. Thing contains some data (attribute) and these attributes can be get and set.

Thing is identified by ThingId. Format of ThingId is:

<fqdn>/<unique string>

Eg.:

- inventory1.sovelto.fi/F26AF3A5-3440-4D54-A6EE-26E49E2400F9
- inventory1.sovelto.fi/T42/Green

Parameters are passed in JSON format:

{ ThingId: "<fqdn>/<unique string>" }

1.4 ATTRIBUTE

Thing has attributes and attribute has value. Attributes are predefined and if other attributes are needed they can be implemented as Extensions.

1.5 ROLE

Roles are used for access rights. Rights are set for each role. Roles are predefined, you cannot create new roles. You can just give rights for each Role and set Role Members to each Role. All available role names found from:
http://thingtodata.azurewebsites.net/api/Metadata/EnumValues?enumName=Role

1.6 RELATION

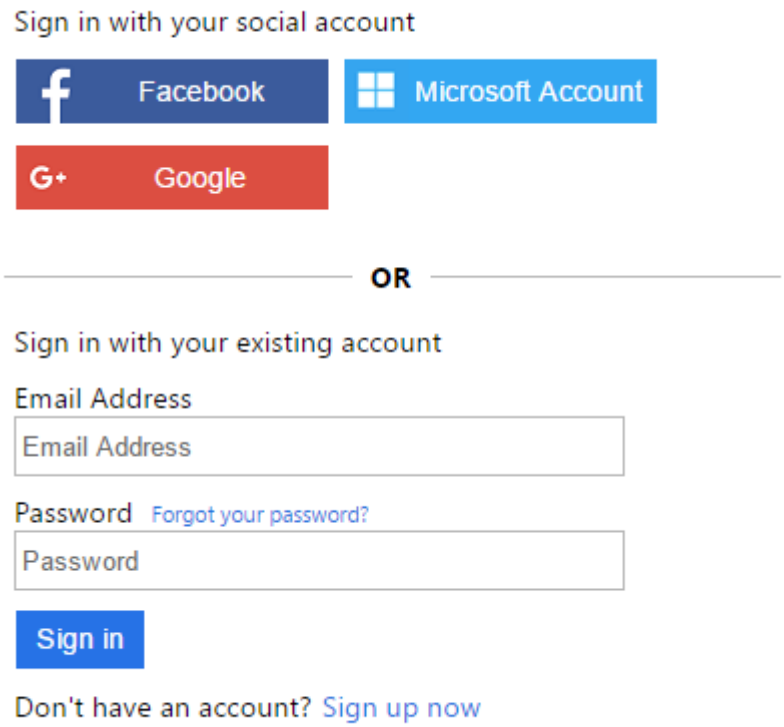
Relation are used to navigate from one thing to another. Relations are predefined, you cannot create new relation names. Relations do have direction and most of them are in pairs (like Contains and ContainedBy).

Relation RoleIn is a special role. The Thing can require, that Inventory adds automatically Another Thing into this role whenever The Thing has a role in Another Thing. Note: this functionality is not available in current test environment.

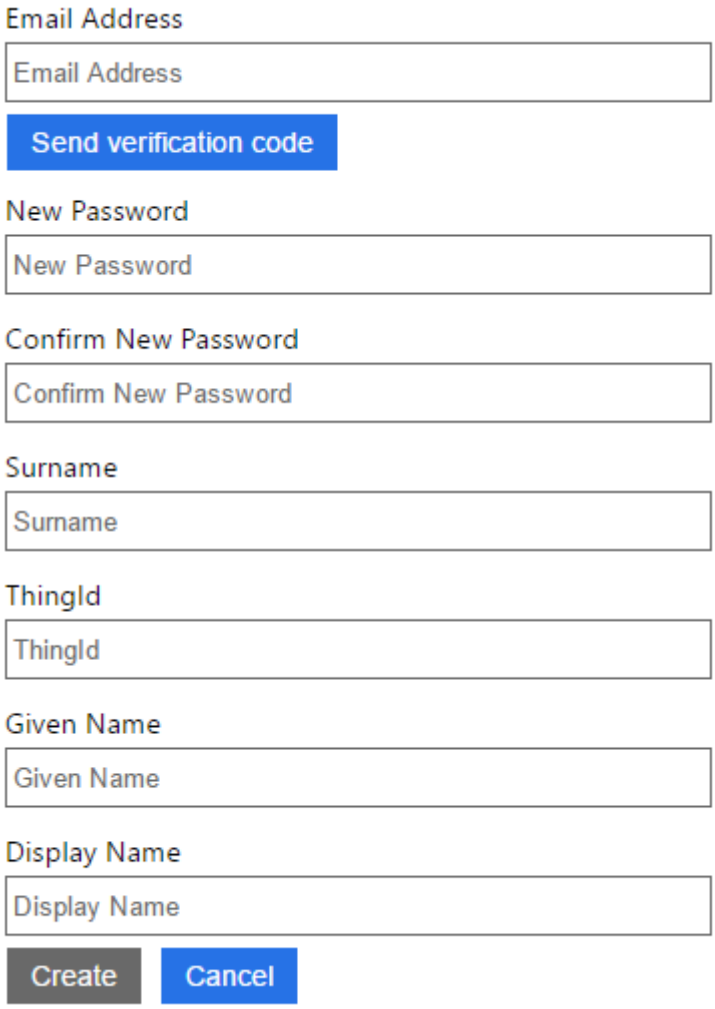
1.7 AUTHENTICATION

Authentication is done by special Authentication Thing. Authenticated Thing is usually connected to a person and her account on some other system, Facebook for example.

To create authentication thing and connect it to Facebook, Google or MicrosoftId user open web site http://thingtodataui.azurewebsites.net/. Click 'Sign in' on upper right corner.



Select proper authentication method and then connect it to thing you want to use as authentication thing. If you click click sign up now you'll form



type valid email address then click Send verification email. Check your mail box and type or copy/paste verification code to field which just appeared to form and then click Verify button. When code is accepted you can enter rest information. Thingid must be valid by format and new thing is created if not found, otherwise sign info is connected to given thingId.

After you have created new login select ShowToken from menu and you'll see access token needed to call EnterAuthenticatedSession API.

Normally when (mobile) application authenticates with Facebook or Google account it automatically gets access token.

1.8 SESSION

Session is needed for almost every API function. Session is for logging and specifying access rights to things. Before you can operate with inventory you have to get sessionId by function EnterAuthenticatedSession or EnterAnonymousSession. SessionId is string and format is GUID.

All things that are accessed during session will be added into Session Access list. When you access any data (create or query attribute or relation) you have to specify in which role this

data is accessed. At least one of the things in your session must have that access right for this data.

For testing and demo purposes there are currently two special sessions identified by ids "00000000-0000-0000-0000-000000000001" and "00000000-0000-0000-0000-000000000002". When these sessionIds are used inventory API makes no security checks. "…1" sessionId is for authenticated and "…2" is for anonymous sessions.

It is strongly recommended that applications use these sessionIds because it makes easier to use T2D API.

1.9 ARCHETYPE

Archetype thing is general thing for special type of thing. For example, "Volvo V70" can be archetype thing in inventory. It can contain general information common to all V70 Volvos (user manuals, chassis type and so on). Specific Volvo V70 car identified by register id XYZ-318 is stored to inventory as (normal) thing.

So, archetype thing specifies the type of thing and thing specifies specific instance of archetype.

There is no inheritance like in object-oriented languages, it is not possible to create new thing based on archetype thing.

1.10 SWAGGER

Inventory API Swagger documentation: http://thingtodata.azurewebsites.net/swagger

2 HOW TO ACCESS API

As ThingToData API is REST API all programming languages and environments can be used if it provides HTTP call and JSON data handling. Swagger site (address mentioned above) can be used for testing API. Also, other tools as CURL or Postman are suitable API testing tools.

Reference implementation of ThingToData API is done with C# and .NET Core and therefor all code samples in this document is written in C# but programming language is irrelevant matter.

3 METADATA AND APPLICATIONS FIRST OPERATIONS

3.1 METADATA

MetaData API contains functions

- ApiVersion
- ApiCompatibility
- **EnumNames**
- **EnumValues**

ApiVersion and ApiCompatibility are not interesting as tutorial so those functions are not covered in this document.

EnumNames function returns list of names (strings) and these are names or definitions which all has set of predefined values. EnumValues function returns values for name passed as parameter.

For example, with GET request URL
http://thingtodata.azurewebsites.net/api/Metadata/EnumValues?enumName=Attribute

returns all possible attribute names thing can have as data property. All these values are identified by id number. Sample returned JSON data:

```
…{
   "id": 7,
   "name": "Title"
 },
 {
   "id": 8,
   "name": "Description"
 }, …
```

All these name/value lists are needed by every application which uses API. That's why application should call EnumNames ja EnumValues functions at applications startup and store returned values for future use.

4 GETTING SESSION

Before API functions and inventory database can be used application needs session. There are two ways of getting session, authenticated and anonymous. Obviously anonymous session is very limited as accessing things.

Stating anonymous session is done by calling EnterAnonymousSession. This function returns sessionId which is unique guid (eg.: 93626b0c-7fa2-4eb4-965e-390214cfdc55).

Authenticated session needs Authenticated Thing, this thing must be one thing among other things in inventory. API returns sessionId with parameter combination "inv1.sovelto.fi/M100" and "Facebook". M100 is predefined thing in reference inventory and it uses Facebook for authentication.

Test data contains two sessionIds which can be use from applications:

- "00000000-0000-0000-0000-000000000001"
- "00000000-0000-0000-0000-000000000002"

5 FINDING AND USING THINGS

There are three different way to get things from inventory:

- by ID
- by location
- by browsing or navigating from one thing to another using relations between things

5.1 GET THING BY ID

First you need to know id of thing. For example application can read QR code which contains valid id.

Following QR code contains text inv1.sovelto.fi/T1 and can be used as testing purposes. Anyhow, thingId is just a string so it doesn't matter whether application get the id from user typed into input field of or by scanning from QR code or by OCR.

5.2 BY LOCATION

Thing may have its GPS location. API function GetNearbyPublicLocationThings returns GPS locatable things by GPS location. Exact location is not needed, API function uses also distance parameter to find things. You can draw circle on the map center point in given location and radius of the circle is distance. Function returns all things which are has GPS location attribute and is located in this imaginary circle.

Function is
http://thingtodata.azurewebsites.net/api/inventory/Core/GetNearbyPublicLocationThings

and sample body which contains input parameters in json format is something like this:

```
{
  "gpsLocation": {
    "latitude": 60.169856,
    "longitude": 24.938370
  },
  "distance": 10000
}
```

Distance unit is meter. You can also test this function with CURL.EXE:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' -d '{ \
   "gpsLocation": { \
     "latitude": 60.169856, \
     "longitude": 24.938370 \
   }, \
   "distance": 10000 \
 }'
'http://thingtodata.azurewebsites.net/api/inventory/Core/GetNearbyPublicLocatio
nThings'
```

API function returns array of things:


```
{
 "things": [
  {
   "idTitle": {
    "thingId": "inv1.sovelto.fi/ThingNb3",
    "title": "A Thing"
   },
   "distance": 0.499623449903504
  }
 ]
}
```


5.3 BY BROWSING

When you know thingId you can query relations to others things. Let's take thing inv1.sovelto.fi/T1 which is suitcase. Assume that you are user inv1.sovelto.fi/M100 (there is this thingId in test inventory). Then you can query relations from T1 to other things with data:


```
{
 "session": "00000000-0000-0000-0000-000000000001",
 "thingId": "inv1.sovelto.fi/T1",
 "role": "Omnipotent"
}
```


it returns:


```
{
 "relationThings": [
  {
   "relation": "ContainedBy",
```

"things": [

{

"thingId": "inv1.sovelto.fi/T2",

"title": "A Container"

}] }] }

and this means that inv1.sovelto.fi/T1 is contained by inv1.sovelto.fi/T2.

This is the correct way browse from one thing to another thing.

6 USING THINGS ATTRIBUTES

Thing has attributes. Attribute has name and value. Attribute names are defined in API specification and function GetEnumValues returns all attribute names as string array.

Attribute values can be set and get. These values define content or state of the thing.

To set things attribute values call function SetAttribute
http://thingtodata.azurewebsites.net/api/inventory/Core/SetAttributes

with json parameter data:

{

"attributeValues": [

{

"attribute": "string",

"value": {}

}

],

"session": "string",

"thingId": "string",

"role": "string"

}

Several attribute values can be assigned in one time. ThingId defines thing you want to modify.

To get attribute values call GetAttributes and specify string arrays which contains attribute names.

7 CREATE NEW THING

Adding new things to inventory is done by function CreateLocalThing. Model or data for this function is follow:

{

"newThingId": "inv1.sovelto.fi/T42",

"title": "Test42",

"thingType": "RegularThing",

"session": "00000000-0000-0000-0000-000000000001",

"thingId": "inv1.sovelto.fi/M100",

"role": "Owner"

}

Data to create new thing is quite clear, remember that thingId is thing (user) in which context new thing will be created.

8 SERVICE REQUESTS

The thing can provide services. For example, street light pole may have service like 'Bulb burned out, send change request'. When you are walking by the street and see street light pole is not working, you can identify pole by location and then send service request.

Owner of the thing can create services by function CreateService. Services are queryable anonymously, so any application can check what services thing provides if thingid is known.

8.1 CLIENT

Client application can make service request and later ask request status.

First application have to get session, EnterAnonymousSession is probably most used way to get sessionId in this scenario. Then application can send query what services are available for known thing by using function GetServices.

Parameters for this function are session, thingId and role. Role can be 'Anonymous'. Function returns services names as string array, sample:

{

"services": [

"Huollettava, lamppu sammunut",

"Hämäräkytkin/ajastusongelma",

"Remove StreetLight",

```
    "LampunKorjaus"
  ]
}
```

Sample call by CURL:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' -d '{ \
   "session": "00000000-0000-0000-0000-000000000001", \
   "thingId": "inv1.sovelto.fi/SL1", \
   "role": "Anonymous" \
 }' 'http://thingtodata.azurewebsites.net/api/inventory/Service/GetServices'
```

Now all available services are known and service request can be activated by ServiceRequest function. Function parameters are: service, session, thingid and role (anonymous is still valid), sample:

```
{
  "service": "Huollettava, lamppu sammunut",
  "session": "4213f54a-91bb-43b1-402d-08d4a373585e",
  "thingId": "inv1.sovelto.fi/SL1",
  "role": "Anonymous"
}
```

Post request returns 200 if service was activated successfully, otherwise return code is 400.

Session is important because you can ask what are service requests statuses by session. Function GetServiceStatus returns statuses, parameters are serviceId, session, thingId and role. Actually, at this moment you to not know what is the serviceId for the service you just requested. Leave this parameter empty and function returns last 10 statuses. Sample return value:

```
{
  "statuses": [
    {
      "serviceId": "7e221783-0bd6-4ab2-2039-08d4a5b63a67",
      "title": "Huollettava, lamppu sammunut",
      "requestedAt": "2017-05-28T10:55:34.6474291",
      "state": "NotStarted"
    },
```

Possible state values are: NotStarted, Started, Done, Failed and NotDoneInTime.

Application must store sessionId so that status query can be done later.

8.2 SERVER

Term server is not accurate in this context, here it means all activities to create services and change the status of the requested services.

First, create new service for specific thing by function CreateService. Parameters or actually json data is:

```
{
  "title": "string",
  "mandatoryActions": []
  "optionalActions": [],
  "selectedActions": [],
  "pendingActions": [],
  "session": "string",
  "thingId": "string",
  "role": "string"
}
```

Actions are defined as data structure:

```
{
    "actionType": "GenericAction",
    "title": "string",
    "alarmThingId": "string",
    "objectThingId": "string",
    "operatorThingId": "string",
    "timespan": "string"
  }
```

Explanations

actionType: current API version supports only GenericAction

title: Name of the service

alarmThingId: thing which will get alarm message if service is not completed on time

objectThingId: thing to which this action should be done

operatorHingId: thing which process this service

timespan: In which time this Action should be done

mandatoryActions: These action must be executed

optionalActions: at least one of these actions must be executed

selectedActions: Exactly one of these actions must be executed

pendingActions: ?

Services are processed by operatorthing identified by thingId (operatorThingId). Operator can query requests by function GetActionStatuses and its parameters are session, thingid and role.

Sample:

{
"session": "4213f54a-91bb-43b1-402d-08d4a373585e",
"thingId": "inv1.sovelto.fi/M100",
"role": "Owner"
}

Function return services assigned to given operator, sample:

{
"statuses": [
{
"actionId": "39f5b85f-70e7-4e21-3b25-08d4a5b63a70",
"title": "Change bulb",
"addedAt": "2017-05-28T10:55:34.6474291",
"state": "NotStarted",
"actionType": "Mandatory",
"actionClass": "GenericAction"
},

If serviced is known function GetActionStatus can be used and it returns more detailed information of the service (service and actions are synonyms).

When action (service request) has been processed, status can be updated by function UpdateActionStatus. Valid status values are: NotStarted, Started, Done, Failed and NotDoneInTime.

9 EXTENSIONS

Thing contains pre-defined set of attributes. If there are other data thing should have in inventory extensions can be used. Extension is actually thing which looks like attribute. Extension attribute data is just a string. The Client Application is fully responsible for all business logic of the data.

Test data for Hackathon contains extension "inv1.sovelto.fi/Fiskars". When application queries things attributes by GetAttributes and with json:

{

"attributes": [

"Title",

"inv1.sovelto.fi/Fiskars"

],

"session": "00000000-0000-0000-0000-000000000001",

"thingId": "inv1.sovelto.fi/SolidTrowel",

"role": "Omnipotent"

}

return value is:

{

"attributeValues": [

{

"attribute": "Title",

"isOk": true,

"errorDescription": null,

"value": "Solid Trowel"

},

{

"attribute": "inv1.sovelto.fi/Fiskars",

"isOk": true,

"errorDescription": null,

"value": "{\"Activity\":\"Cultivating\",\"BusinessCategory\":\"Garden and Yard Care\",\"CountryOfOrigin\":\"FI\",\"DateOfManufacture\":\"\\/Date(1327795200000+0000)\\/\",\"EanCode\":6411503460101,\"Manufacturer\":\"Fiskars\",\"MarketArea\":\"Reg. Pan-

European\",\"ProductDescription\":\"Solid
Trowel\",\"ProductLine\":\"Solid\",\"ProductPage\":\"http:\\/\\/www.fiskars.eu\\/products\
\/gardening\\/nursery-tools\\/trowel-1000694\",\"ProductType\":\"Planters\"}"

}

]

}

Title attribute is quite obvious. Fiskars-extension contains data as json format like ProductType and ProductPage.