# Thing2Data introduction

# This introduction describes full API-functionality of Thing2Data -reference implementation not including all maintenance related features. Initially only Thing2Data Lite -version will be available. Available features can be checked from Swagger documentation.

## 1 A BRIEF INTRODUCTION TO THING2DATA ONTOLOGY

Thing2Data is a cloud service. It is a database of things – somewhat similarly to Facebook. Facebook contains data on us, which data can be maintained by us and accessed by others. Thing2Data maintains data on anything that can be given an individual identity.

Facebook has an owner. Thing2Data can have multiple owners as it is a collection of multiple databases. Each database is called an Inventory. Thus we have individual items, whose data is stored in one of the Inventories, which can be set up and maintained by anybody.

Thus, everything starts from individual items. They can be contracts, bus-stops, rooms, chairs, cars, food packages, trees, seats or you name it. They can be material or immaterial if only they can be individualized.

These individual items can be given a Thing2Data identity and a database object in a Thing2Data Inventory. We call that database object a Thing. The identity is an address to the Thing. So that we would not need to remember all identities it is possible to tag the individual items with tags that contain the whole Identity or a partial Identity.

For example a car has a license plate 123-ABC. It may act as a tag and we might deduce its identity as CarInventory\123-ABS. The corresponding Thing CarInventory\123-ABC most probably resides in Inventory named CarInventory unless it has been transferred to some other inventory but in this introduction we should stick to basics.

Each Thing contains information. A Thing has a name, it has location, and various other attributes. Some of these refer to the corresponding items physical properties and others to the relations it has with other Things. A Thing can i.e. be a part of another Thing. It can give permission to other Things to read or write it's attributes.

As each Thing is a separate entity and all Thing attributes are in its own control, it should be considered perfectly normal for example if Thing A would claim to be a member of Thing B, but Thing B would not consider Thing A as its member. In general, there is no authority, who could decide, which Thing's claim is correct. Each Thing so to say, has a right to its opinion.

An important attribute of a Thing is the Service Request registry. Each Thing can define one or several Service Request -types. If a Service Request is activated, the Inventory informs other Things if they are required to perform some action on behalf of the requesting Thing.

For example if the Thing CarInventory\123-ABC would be wrongly parked, one could possibly activate a AlarmOwner -named Service Request, which would deliver an Action Request to the owners mailbox Thing.

There are several different Thing-types. The most important are the following.

Creator Thing is allowed to create other Things and corresponding identities in the Inventory where each Creator Thing resides. Creators are granted access and authenticated by the Inventory.

Authenticated Thing is usually connected to a person and her account on some other system, Facebook for example. In this case I could connect to my own authenticated Thing and it would ask my Facebook account to confirm that I have logged in to Facebook.

Regular Thing is as the name implies, used for common purposes. It can store a variety of information and relations about the actual item whose information it contains.

## 1.1 How to use the Inventory database

Let's assume that we have an Inventory full of Things.

I open my Thing2Data application and start a session logging in via my FB Authenticated Thing. This phase can be automated and further examples equate the user with his Authenticated Thing. This however forms the basis of logging and security.

The next phase may be that I use my application and navigate through all Things that are somehow connected to me. This would be similar to many contemporary applications. But more common with Thing2Data is to use the tags or location information and select individual items by pointing at them.

If the tag or position yields a recognized Thing Identity, the Inventory performs a check if I have been given rights to access the Thing I pointed out. Then I can read or update the information or navigate further to other Things related to the pointed Thing. And naturally I can also invoke all Service Requests related to this Thing.

All operations are handled via simple or more flexible applications that can be coded to all common mobile platforms using universal REST API.

## 1.2 Thing types – a more detailed list

A Creator Thing is provided by each Inventory as an authenticated access point for each such party that is allowed to create Thing Templates for regular users from their unique ID-space.

A Thing may be Archetype if it refers to an unlimited number of similar copies. General information of a serially manufactured item or a digital file may be examples.

A Regular Thing refers to an individual instance, which may be a physical thing or a place, a contract or other physical or digital object, which has individual attributes.

An Alias Identity Thing refers to an otherwise empty Thing that in it's turn refers to another thing via Alias Identity relation. Aliases can be used for security purposes, for example as passwords or certifications of authenticity or plainly as synonyms for the Thing Identity.

An Authenticated PID Thing refers to a known AUID-Type and Inventory validates the accessing application using the known AUID-Type specific method.

An Archived Thing refers to a former Regular Thing whose Template has been transferred to another Inventory. An Archived Thing is visible only to a Session, which specifically asks for it and it can only be used to entitle access to other local Archived Things.

An IO Thing refers to an IoT-device or a Bot, whose information may be linked to a Regular Things Attributes. IO Thing contains the description on how to interact with the Device/Bot.

A Wallet Thing refers to an electronic wallet, which may be used by a Regular Thing for monetary transactions – paying and receiving payments. It contains the description on how to interact with the appropriate payment system.

## 1.3 THING – THING NAVIGATIONAL RELATIONS

Thing-Thing relations are attributes that point to other Local or Public Things. Relations have types to indicate different relationships and their directions. A Thing can i.e. belong to another Thing as a logical member or be contained by it physically or be derived from it. Roles, which are described below, are also used to indicate relations in a more flexible manner. Relations are unidirectional and they can be used to navigate Thing2Data Inventory. The Thing that is being referred to may however not grant access. Only Roles can supply access rights and it is up to the referred Thing what access Rights it grants and to whom.

## 1.4 THING DATA ACCESS ROLES

Access roles are predefined groups such as Owner, Administrator, Member, etc., which can be used to form groups. All such groups are attributes of the Thing that supplies the roles. All passenger Things can i.e. be added to Member or User role in a vehicle Thing. This way each passenger Thing gets access rights defined for Member or User role in the vehicle Thing. Some of the predefined Roles affect the way how an Inventory handles them.

## 1.5 THING ACCESS RIGHTS

Thing access rights are CRUD-type rights given to Access roles. Each Attribute including each role list should be considered separately when necessary. This means for the previous Passenger example that any Member Role member might be given a right to read the Identity of the Driver and Location data, but not for example Identities of other Members.

## 1.6 SERVICE REQUESTS

Mainly Thing2Data Inventory is a passive data warehouse and mainly it is the task of applications to change the content of Thing attributes. However, there are occasions where message delivery should be automated due to a wish to avoid extensive polling or revelation of private information contained in roles and relations. A flexible Service Request system is

used to deliver timed or immediate notifications and their responses from Things to Things. A simple notification service is predefined for all Things.

Service Request types are defined separately for each Thing. The syntax contains ordered and unrestricted, obligatory and alternate Actions and specific Wallet- and IoT- Actions. Service Request type and its Actions are named by descriptive strings. A service request, when activated is placed in the outgoing service request queue of the Thing. Each Action refers to the Thing, which is asked to perform the Action by placing the Action Request in its incoming Action Request Queue. Each Action may also contain a timed alarm and ID of the Thing who is recipient of the failure message. Completed Service Requests are so marked and Service Request Queue acts as a Log for completed Service Requests.

## 1.7 THING DATA EXTENSION SYSTEM

Predefined Thing attributes are not sufficient to cover very specific applications. Thing2Data supports an Extension system, where each Extension type contains a data structure, which follows the definition of that Extension. Extension types and their specifications may be published by Inventories and www.thing2data.com web pages.

## 1.8 IOT-DEVICES, BOTS AND WALLET DEFINED VIA IO THINGS AND WALLET THINGS.

IoT-devices can be linked to Things to provide Things with automatically updated data i.e. location, temperature etc. Applications such as software robots – bots in short – can be linked to Things to provide automated algorithmic or AI type functionality to Things. One example is wallet functionality through which a Thing can arrange payments and receive them. The required special software for IoT-devices or monetary transactions is not part of Thing2Data Inventory but it needs capability to interact with Thing2Data Inventory API and proper access to the Thing it is required to serve.

## 1.9 THING ATTRIBUTES LIST (PARTIAL)

- Thing Identity
- Thing Type
- Thing – Thing navigational relations list
- Role to access rights mapping
- Role ID list
- Current location
- Thing Status
- Service log syntax & service types
- Requested (from roles) services que with triggers
- Authored Log of performed services
- Physical properties syntax & list
- Digital properties syntax & list
- List of extensions in use
- Thing wallet information
- IOT&Bot information
- List of Language Extensions

# 2 INVENTORY SPECIFIC DATA STRUCTURES

## 2.1 INVENTORY URI & T2DBC

Inventory URI is fixed and unique to this Inventory and is attached to the beginning of all Thing ID:s that this inventory publishes to T2DBC.

Inventory maintains T2DBC-address-information and also information on the Trust-status it has with T2DBC.

## 2.2 INVENTORY VERSION & COMPATIBILITY

Inventory maintains the T2D-API version number and a list of supported API-calls

Inventory maintains a list of compatible API-calls for each previous version

## 2.3 LIST OF TRUSTED INVENTORIES AND LIST OF INVENTORIES THAT TRUST THIS INVENTORY

Inventory maintains a list of other Inventories it Trusts

Inventory maintains a list of other inventories that Trust it

This Trust may be established via Blockchain if possible.

## 2.4 LIST OF TRUSTED CREATORS

Inventory maintains a list of Trusted Creators, who are allowed to use CreateLocalThing – function. For symmetry purposes, each Trusted Creator is provided access to corresponding Creator-type Thing.

## 2.5 LIST OF AUID-TYPES AND RESPECTIVE AUID-DRIVERS

Inventory maintains a list of supported AUID-Types and respective AUID Drivers in order to support authentication of access to AUID-Type Things. AUID-Thing cannot be created or used as an Entry-point or otherwise accessed without permission from AUID Driver. AUID Driver must have internal knowledge on respective Authenticating server.

## 2.6 SESSION LOG

Optional External SessionID, Remote Inventory URI,

(Local) Session ID & Token

Entry-point Thing ID

Session language

List of Accessed Thing-Role -pairs

# 3 THING ATTRIBUTES NONTECHNICAL DESCRIPTION

## 3.1 THING IDENTITY

### 3.1.1 Thing Title

### 3.1.2 Thing Identity

### 3.1.3 Creator ID

### 3.1.4 Created timestamp

### 3.1.5 Published timestamp

### 3.1.6 Most recent change timestamp

## 3.2 THING TYPES

### 3.2.1 Creator Thing (CRID)

### 3.2.2 Regular Thing (THID)

### 3.2.3 Archived Thing (AHID)

### 3.2.4 Archetype Thing (ARID)

### 3.2.5 Alias Identity Thing (ALID): Secret/Normal

### 3.2.6 Authenticated Identity Thing (AUID)

### 3.2.7 IO Thing (IOID)

### 3.2.8 Wallet Thing (WAID)

## 3.3 THING-THING NAVIGATIONAL RELATIONS

### 3.3.1 Thing1 contains Thing2

### 3.3.2 Thing1 is contained by Thing2

### 3.3.3 Thing1 inherits archetype Thing2

### 3.3.4 Thing1 originates from Thing2

### 3.3.5 Thing1 has a role in Thing2

- automated maintenance, possibility to reject by destination

#### 3.3.6 Thing1 ID has Thing2 as an Alias

#### 3.3.7 Thing1 has a Wallet Thing2 (ordered one to many – A Thing may have several Wallets at its disposal and there needs to be at least two categories available – prioritized and secondary wallets)

## 3.4 THING ACCESS RIGHTS BY ROLES

### 3.4.1 Role to access rights mapping, general description, Each IO Thing is a role by itself

Access Rights Table –structure is described. Each Role and Each Attribute form a table where CRUD-rights are granted. Omnipotent, Anonymous, Alias, ArchetypeMember and CurrentVersion –roles may have special meaning through added automatic rights or other special procedures. Others are generic in the way Inventory handles them.

Access Rights Table or Role Member List includes logging requirement and local access – requirement!

Besides roles, each attribute can be linked to one IO Thing, if the IO Thing and the Attribute in question are required to be synchronized via T2D Inventory!

### 3.4.2 List of Attributes in the Access Rights Table

At present, it is assumed the list is static metadata and there are no specific functions defined in API to edit or read this table.

Attribute list should be described here: XXX

### 3.4.3 List of Roles in the Access rights table

**Generic Roles**

- Owner
- Administrator
- Member
- User
- Maintenance
- Logistics
- Viewer
- Manufacturer
- Gatekeeper1
- Gatekeeper2

**Special Roles**

- Omnipotent
- Anonymous
- Alias
- ArchetypeMember
- CurrentVersion

## 3.5 THING ACCESS ROLE MEMBER

Access Rights Table and Role Member List both include potential logging requirement and local access only –option!

### 3.5.1 Role Member list structure

Role; Thing Identity-list; Logging; Authentication Local

### 3.5.2 Logging and Authentication requirement for each Identity

If Logging is true for CRUD, each respective access is logged to RoleAccessLog.

If Authentication requirement is Local, access is denied if permission is required by a Thing in Role-list that is not Local.

## 3.6 SESSION IDENTITY LOGGING OF ROLE BASED ACCESS

### 3.6.1 For each Role that has Logging defined

RoleAccessLog structure: Role: Session ID, Attribute, access-type, timestamp

## 3.7 THING LOCATION

### 3.7.1 Current location type

GPS/Street/ID

Fixed/Stationary/In Transit

### 3.7.2 GPS Coordinate

GPS-coordinate

Time Stamp

Public Coordinate

### 3.7.3 Street Address

### 3.7.4 ID of another Thing

### 3.7.5 Preferred location

## 3.8 THING STATUS (SELECT ONE)

#### 3.8.1 Template

#### 3.8.2 Destroyed/Divided

#### 3.8.3 Planned

#### 3.8.4 Normal & operational

#### 3.8.5 Waiting for service

Malfunctioning

Missing

Danger

Available for rent or sale

### 3.9 SERVICE LOG SYNTAX & SERVICE TYPES

#### 3.9.1 Service types registry

ActionRegistry

ServiceRegistry

#### 3.9.2 Service Request Types general syntax

ServiceTitle

MandatoryActionList

OptionalActionList (you may perform 0-n of these actions)

SelectedActionList (one is required – and enough)

PendingServiceList (next Service Requests to be started including their ThingID's)

ServiceStatus

AvailableforRoleList

#### 3.9.3 Action Type syntax

Action Title

Action Type: Generic / PaymentRequest / ReceiptRequest / IoTBotRequest / ServiceRequest

Requesting Thing ID

Bidden Thing ID

Bidden ServiceTitle

Payment/Receipt: Payment reference, WalletID, Currency, SumValue, MaxCommission

Status (waiting, completed, failed, canceled)

Deadline

Alarm ID

BlockChain (this is needed if we allow some Actions to be committed to T2DBC.

## 3.10 Outgoing Service Request Queue

Service Request Title, SessionID, timestamp, initiator, status; Mandatory Actions: Action Title, Action Type, BiddenID, Bidden ServiceTitle, Payment/Receipt, Status, Deadline, AlarmID, Status; Optional Actions: Action Title, Action Type, BiddenID, Bidden ServiceTitle, Payment/Receipt, Status, Deadline, AlarmID, Status; PendingServiceList, ServiceStatus

## 3.11 Incoming Action Request Queue

Action Title, Session ID, Action Type, timestamp, Requesting ID, Bidden ServiceTitle, Payment/Receipt, Status (waiting, completed, failed, canceled), Deadline

## 3.12 Authored Log of performed services

*Inventory Logs the services a Thing has requested by appropriately changing the status of the Service Request in the Queue*

*Inventory Logs the Actions performed or failed by changing the status of the Action Request appropriately in the Action Request Queue*

## 3.13 Physical properties syntax & list

CN-code (6 numbers, Harmonized Commodity Description and Coding System, World Customs Organisation)

EAN Code

Weight kg

Volume liters

ContainerVolume liters

MaxSurfaceDimensions x,y,z

FreeContainerDimensions x,y,z

AllowedTopDirection x,y,z (check all appropriate options)

MaxFluidLevel

CurrentFluidLevel

LoadabilityOnTop kg

LoadabilityWithin kg

Luggability easy/requires packaging/requires extra care/security considerations/hygienic considerations (check all that apply)

NumberLooseParts

InsideTemperatureRange (minimum, maximum)

CurrentInsideTemperature

OutsideTemperatureRange(minimum, maximum)

CurrentOutsideTemperature

Weatherproofness no/moderate/good/excellent

MaxGForce

Color

AttentionDate

ObjectActivityClass (biological, chemical, electromagnetic, robotic, alive, neutral)

HazardousCargo boolean

SpecialConsideration (text)

IoTField1
-title, SI-unit, Data

IoTField2
-title, SI-unit, Data

IoTField3
-title, SI-unit, Data

## 3.14 DIGITAL PROPERTIES SYNTAX & LIST

Location URI

Protocol/API

Key

DocumentFormat

DocumentSize

MinimumBandwidth

ApplicationPlatform

Language

LengthAV

NumberofPages

SecurityProtocol?

## 3.15 THING DATA EXTENSION SYNTAX & LIST

Currently Function in Thing Story 26 expects Extension to be a block of data, whose syntax is known to applications and defined in Thing2Data.COM. This requires a specified Role/Access-rights list in the beginning of the block.

## 3.16 LANGUAGE EXTENSION

Mutilingual Things are implemented via Language Extensions. A list of available language extensions is required, which should contain a default language. Inventory automatically returns attributes from the suitable Extension for attributes containing language dependent information when language switch is used.

## 3.17 THING WALLET DATA

Wallet Thing contains necessary attributes for setting a Thing up with Navigational relations, Roles & Access Rights and accessing it in a controlled fashion. Service Transaction support is also useful. Only following extra attributes are necessary for performing its function:

List of allowed currencies

Maximum daily payment??

Wallet-type

Account number

Key

## 3.18 IOT-DEVICES AND BOTS DEFINED VIA IO THING

IO Thing contains only necessary attributes for setting a Thing up with Navigational relations, Roles & Access Rights and accessing it in a controlled fashion. Service Transaction support is also useful. Only following extra attributes are necessary for performing its function:

IoTBotURI

IoTBotKey

SupportedAttributeList

SupportedActionRequestList

### 3.19 VERSION RELATED DATA DEFINED VIA EXTENSIONS

# 4 LIST OF THING STORIES – CORE LAYER STORIES AND APPLICATION STORIES

## 4.1 SETTING UP A THING2DATA INVENTORY AND CONNECTING IT TO T2DBC

1. Setting up a Thing2Data Inventory with T2DAPI (Core 1)
   Setting up such a service, which implements Thing2Data API, and allows at least one user to create Things with unique Identities which can be accessed via T2D API.
2. Authorizing an URI to create Local Things (Core 1)
   An Inventory must provide means to create Local Things. Inventory must provide means to guarantee Local uniqueness of the Identities by itself or require that Identities contain URI of their Creator and require Creator to guarantee the uniqueness of Things they create. There is no mandatory requirement for creator URI to be part of Local Identity. But Inventory URI must always be a part of Identity even when it is Local only.
3. Enabling a T2D-Inventory to bond with T2DBC (Core 2)
   Thing2Data Inventory must bond with Thing2Data Blockchain in order to Publish Identities of its local Things. Content of this bonding requires further study, but it may mean that the inventory declares itself and possibly gets some public support and possibly takes part in Blockchain calculation.
4. Declaring trust between T2D-Inventories (Core 3)
   An Inventory may ask another inventory to trust its Things. This request is initiated by a Local Thing in an Inventory asking to trust Things in another Inventory. It is recommended that Inventories routinely trust each other and only deny trust when they have clear reason to do so. This may be the case for example when an Inventory has a faulty T2D API or fails to meet security standards. Things that require stricter security standards than exist between Inventories, can require Local Authorization.

## 4.2 CREATING A THING TO AN INVENTORY AND ACCESSING THE DATA OF A THING

5. Authorized URI creating a Local Thing (Core 1)
   Using T2DAPI to create a Local Thing. Inventory provides authorization by its own means to a known User, who may be identified by URI and other means. This user may later be called Creator type Thing.
6. Publishing Local Things into Public Identity Things (PID) (Core 2)
   Anyone with Omnipotent access rights to a Thing may ask an Inventory to Publish a Local Thing into a Public Identity Thing. The Identity must contain the Issuing Inventory's URI. When Publishing a Thing the Inventory must be bonded with T2DBC and it must register the Published Thing Identity into T2DBC. Publishing Inventory must know it has not previously Published a Thing with an identical ID.
7. Anonymous viewing and editing of a Thing (Core 1)
   A Thing can allow anonymous user in a role and give this role access rights. This way an application can access a Thing without providing anything else but the Thing's Identity (and application session). Viewing and editing takes place via T2D API.
8. Transferring a Thing from one Inventory to another, accessing Archive Thing (Core 3)

If an application that has access to a role with full rights to a Thing's Identity asks to move a Thing to another Inventory, the current Inventory requests the receiving inventory to set up a Thing with an identical Identity. The Thing in the originating Inventory is converted to Archived type Thing and the originating Inventory registers the receiving Inventory as the new site of the Thing (Identity). The old version remains an Archived Thing, unless deleted and the application can transfer any such information it needs and has access to, to the new version.

9. Creating an Authenticated Identity Thing (AUID) (Core 3)
   This feature is implementation specific. AUID-type refers to a known Authorization method, through which Inventory validates the accessing applications right to access the Authenticated Identity Thing. This method is used i.e. if an Inventory supports third party authorization and identification systems. For example Facebook identity can be used so that the Inventory would create an AUID Thing and connect it to a specific persons Facebook-account. AUID would contain the Inventory name, AUID-type and FB-account ID.
10. Creating an Archetype Thing to an Inventory (Core 2)
    Creating an Archetype Thing takes place much the same way as creating a Regular Thing except setting the Thing type into Archetype. This Thing story describes how to configure roles in such a way to make Archetype Thing useful as an inheritable model.

## 4.3 CONTROLLING ACCESS TO THING2DATA

11. Setting up role based access rights for a Thing (Core 1)
    A Thing can only be accessed through belonging to a role that allows access. Access roles form a table. Each role is given or denied CRWD-type access to each Thing attribute. When accessing an attribute, the right to it is checked against this table.
12. Setting up role member lists (Core 1)
    Each role for a Thing is populated by listing Identities of Things that belong to a role when accessing the Thing in question. When accessing a Thing in a role, the right to use the role is checked against this list.
13. Connecting an Alias ID to a Thing (Core 2)
    A Thing may be used only to point to another Thing with no content or meaning of its own. Alias Identity Thing is created like a regular thing and its Type is set to Alias and a pointer created to the actual Thing ID. The actual Thing yields all permissible access rights to itself through the Alias.
14. Authenticating Identity using alias ID or Issuer password (Core 3)
    When an Alias Identity Thing (ALID) hides its relation to the actual Thing, but is still grants a role to the Alias Identity Thing as the only Thing with access to Access Role list, application needs to know both the Thing ID (analogous to user name) and the Alias ID (analogous to password) to gain access to the Thing. When Authenticated User ID Thing is used as the Entry-point, collaboration between the authentication server and Inventory provide authentication by their own defined means.
15. Setting up a person or a list of persons as a Thing (Core 3)
    A person can be described as a Thing that has as its members different AUID Things and/or Things that use Aliases as passwords. Thus a person is a collection of that person's access methods. The person as a Thing can be given a role in accessing other Things when there is no requirement to differentiate between access

methods. Similarly, an organization can be defined as a Thing that has other Things as its members.

## 4.4 NAVIGATING THING2DATA VIA ROLE RIGHTS

16. Inventory listing of my Things (Core 2)
    T2DAPI supports a list of Things where a given thing has a role or a relation. The application needs access right to such information and role information is optional. The simplest application is to access a Thing and list other Things in which it has a Role or a Relation and do this Role by Role and Relation by Relation. This assumes that the accessed Thing has opted that the Inventory automatically maintains the list of other Things where it has a Role.
17. Reading or updating data of a Thing via another Thing (Core 1)
    A Thing is accessed and it has a Role in another Thing. Using the accessed Things information, the second Thing is accessed in the Role that the originating Thing is allowed to. The second Things Attributes are read or updated.
18. Inheriting data when a Thing is transferred, divided or assembled (Core 3)
    When a Public Thing ID is moved to point from one Inventory to another, or a new Thing is separated from another or a new Thing is assembled from previous Things, the previous Things may become outdated but still necessary for archive purposes. There also needs to be a way to read their data in case the new Thing has access right to it.
19. Inheriting data when a Thing refers to an Archetype Thing (Core 3)
    When an Archetype thing is created, it may be typical to give access rights to read the attributes to anyone or to regular Things that inherit the Archetype.

## 4.5 DEFINING AND ACCESSING SERVICE REQUESTS

20. Anonymous notification as a service request (existence) (Core 4)
    One of the simplest useful functions of Thing2Data is when an application accesses a Thing and activates a predefined Service Request, delivering a Service Action Request to another Thing. If for example some physical thing is lost, its owner may allow setting its respective Thing Location Attribute anonymously and activate Service Request as a means to notify the owner without knowing who the owner is or getting any other information from the Thing but its Identity.
21. Defining Service Requests for a Thing (Core 4)
    Service Requests are defined using a specific syntax and stored as available Service Request-types waiting to be activated. Service Requests are usually defined by the owner and activated by various roles. When a Service Request is activated by an application acting in a role that permits the activation, Inventory places the Action Requests listed in the Service Request to the Action Request Queue of each Thing respectively that is requested to perform Service Action according to the Service Request definition.
22. Performing and logging Service Requests (Core 4)
    When noticing an Action Request in the incoming Action Request Queue, an application is supposed to perform the requested Activities as it sees fit and then confirm the performed Activities as completed or failed in its own Action Request Queue. Inventory confirms the Activity then completed or failed in the Service

Request Queue of the requesting Thing and time stamps and confirms the Activity Request in the Things own Activity Request Queue. The Inventory checks after each Confirmation if the whole Service Request of the Requesting Thing has been performed and if so, it timestamps and confirms the Service Request to Service Request Log. It also cancels extra Optional unperformed Activities. Inventory also monitors scheduled alarms and handles the issue when an Activity has not been performed in due time. This method is defined later.

23. Reading and purging Service Request Log (Core 4)
    Service Request may be deleted from the Queue before or after logging when acting in a role that allows it. A Thing that has serviced a Service Request Activity has its own confirmed copy of the action in its own Service Request Queue and does not usually need access to the serviced Things Log to prove confirmation of Action.

## 4.6 CONFIGURING THING2DATA APPLICATIONS AND INSTALLING NEW IDENTIFIER-TYPES

24. Installing Thing2Data –ready applications (Core 2)
    Thing2Data applications are installed just like any other application. The T2DBC-address should be configurable and direct access to a known Inventory should be possible.
25. Installing new Identifier Type Recognizers and accessing nearby identities based on their location (Core 3)
    Applications should have installable Identifier recognizers. Application should ask or enumerate the recognizers and ask manual help if recognition fails or is partial – like in the case of car license plates, EAN-codes or visual and GPS-recognition.
    Thing2Data supports identification of nearby Things if their GPS-position is defined as Public. This information is provided separately by each server and can in principle be consolidated to cover all inventories by any 3rd party operator.

## 4.7 CONFIGURING AND USING EXTENSIONS, IOT-DEVICES AND BOT CONNECTIONS

26. Accessing Extension data of a Thing (Core 4)
    In addition to predefined attributes, Thing can also contain Extensions as Attributes. Allowed Extensions are listed and each Extension is structured in a manner specific to that Extension. Extensions may contain information for customs, medical information, specific logistics information food, medicine, or chemical information etc.
27. Linking a Thing to a Bot or an IoT –device (Core 5)
    A Thing may be connected to an IoT-device or a software robot that provides data to linked attributes or performs requested actions. These IoT-devices or Bots are configured as IoT-type Things and links are configured to the appropriate Thing attributes that are to be maintained by the mentioned IoT-device or Bot.
28. Having a Bot or IoT-device reading & updating data for a Thing (Core 5)
    The Bot or IoT-device can communicate with the Thing like an application through T2DAPI and if so, must supply an Identity, which allows it a Role with suitable access rights. If extra security is required, AUID Things should be used. However this is not an efficient way to connect a large number of IoT-devices. For them, when they are linked to any attribute in the manner described in Thing-story 27, the Inventory automatically updates the attribute from the IoT-device or Bot when the attribute is

accessed. It also automatically delivers Action request to an IoT-device or Bot, when an activated Service Request includes an Action that is to be delivered to an IoT-type Things Action Request Queue.

## 4.8 CONFIGURING AND USING A THING-WALLET

29. Setting up a wallet for a Thing, configuring wallet-extension (Core 5)
    In order for Thing X to be able to receive payments and pay for requested services, the Thing may be linked to an electronic wallet. Wallet is configured as a Wallet Thing, which gives appropriate Access Right to Thing X and it is connected to the Thing X through Navigational Wallet Relation. Wallet Thing contains account type information, account number and key, usage limits and access right information.
30. Receiving payment from and paying to a Thing (Core 5)
    A Thing can request payment via Service Request Queue using a specific activity for requesting payment and requesting confirmation for received payment. Payment is accomplished when Payment Action Request is acknowledged. All payments are from a Thing to another Thing, but on deeper level from one Things Wallet to another Things Wallet and still deeper level, from one Wallets Account to another Wallets Account. Whoever is required to pay, does not get to know the account or the requestors Wallet ID before actual payment and not even then if the actual transaction system hides the information.

## 4.9 SECURITY & MAINTENANCE

31. Storing & retrieving of Local Session Identity (Core 3)
    When a Session accesses a Thing, supplying another Thing in its Access Path to gain an Access Role, this is logged as a part of the Sessions Access Path. When an Access Role is used to read or modify a Thing, the event is logged if the object Thing requires the used Access Role to be logged.
32. Storing & retrieving of Local Session Access path (Core 3)
    When an application accesses a Thing2Data Inventory, it supplies a Thing Identitity, which is used as the Entry point to a Thing Access Session. If the accessed Thing is available either due to it allowing Anonymous access or it is Authenticated, a unique Session ID is created and the Thing acting as the Entry Point is stored with it into Session Log. If the Inventory supports and requires separate user identifications, they are logged under the Session ID.
33. Using an Access Path from one Inventory to another Inventory (Core 4)
    When the Thing X in Inventory A has a Role in Thing Y, which resides in Inventory B, there needs to be a Trust relationship from Inventory B to Inventory A. In addition, Thing Y must allow remote access (not require Local Access). Thing X must have a Published Identity. Application Accesses Thing Y and provides credentials for Thing Y including Session ID in Inventory A.
34. Storing & retrieving of Remote Session Identity (Core 4)
    When Thing Y is accessed in Inventory B using a role that Thing X has, while Thing X resides in Inventory A, Inventory B initiates a new Session where Entry point is Thing X marked as remote and Session ID in Inventory A attached to it including Inventory A URI.
35. Storing & retrieving of Remote Session Access path (Core 4)

There needs to be a procedure for retrieving the whole Session Access Path, when Session Access Path crosses over several Inventories. This needs to be a part of Trust relationship between Inventories and requires an API. Original Entry-point should have access to the Session path if this can be supported.

## 4.10 User interface

36. Multilingual support (Core 5)
    It is mandatory that dynamic Thing content can be presented in several languages. This is accomplished via Language Extensions. A Language Extension is a data structure that contains such Attributes, which may be language dependent. Each Thing has a default language and if the application requests another language, an appropriate Extension is used if available. If not, the default is returned.

## 4.11 Advanced application examples (+ -marks from a priority assessment of meeting nr 5)

37. Example: Simple maintenance request of a Thing (Core 4 required) ++++
    Thing X is defined in such a way that owner has full access rights to all it's attributes. Owner defines a Service Request called "Maintenance Request" in such a way that when activated, the request is posted to the Owner. Access to add Service Requests is set open to Anonymous users. An anonymous user activates the Maintenance Request, Owner receives it in Owner-Things incoming Service Action queue and marks it serviced. Inventory confirms the completed Service Request.
38. Example: Storage of a Thing including scheduled reminder (Core 4 required) +
    Owner defines a Service Request called "Storage Reminder" in such a way that when activated, the requested Action is posted to Receiver. In addition the Storage Reminder is defined in such a way that it is posted to Owner if the included Action is not serviced before a requested time. After Owner or Receiver services the Service Request, it is confirmed to the Service Request Log by the Inventory.
39. Example: Cold chain assurance service log (Core 4 required) +
    Pickup and delivery of a food basket is defined as a Service Request in such a manner that there are two optional ways to do the delivery safely. One of the options requires the use of cold storage during the wait time between two different transports and the other direct delivery option is required to be fast enough not to require cold storage. The Service Request is defined to monitor the required time limits of total transfer time and of transfer without cold storage. Receiver of the Thing can confirm or fail the Service Request.
40. Example: Purchasing a Thing from itself (Core 5 required) ++
    A Service Request is defined so that Purchaser becomes Owner of the Thing with full access rights and original owner named Seller gets a defined purchase price into its purse from the Purchasers purse. (As a clarification – the actual work here is done by the applications, who confirm the required deeds on behalf of the parties involved. Inventory only guarantees a similar copy of Service Request is logged when completed)
41. Example: Renting a Thing from itself +++++
    A Service Request is defined in such a way that Lessee gets suitable Access Rights to use the Thing after the Rent is paid to the Lessors Purse or the Rented Thing's Purse. (As a clarification – the actual work here is done by the applications, who confirm

the required deeds on behalf of the parties involved. Inventory only guarantees a similar copy of Service Request is logged when completed)

42. Example: Transporting a Thing – including service request and payment +
    A Service Request is defined, which actuates pickup, delivery, receipt of delivery, payment of delivery and time constraints.
43. Example: Using an EAN Archetype Thing Identity as a private Thing
    An Inventory may grant an URI authority to create Local Things. Using this option, user's application can add Inventory URI and their own URI in front of EAN Achetype ID to define a Local Thing, which can be recognized by the user herself by the EAN. In case there are several things with the same EAN, enumeration or time stamp can be used if they need to be separately identified.
44. Example: Using a Thing Identity and it's Alias as User ID & Password
    When an Alias ID Thing is the only one having any access rights to a Thing, the Thing Identity can be used as a User ID and Alias ID as the corresponding password. In order for this to be secure enough, the Thing Access Role Member –list should not reveal the Alias ID. Either the list should be encrypted or Alias ID should always be encrypted when stored in that list. This is to provide added security against hacking as in this case, the Alias ID Thing is the only one with access to the list. Also, as a precaution, Alias ID Thing should not be logged as the Entry point in Session Log, but the accessed Local Thing instead.
45. Example: Service & maintenance of a Thing by a group or an organization ++
    A Service Request named Maintenance Request is defined in such a way that it is posted to a Thing, which represents a Service Organization. The said Thing gives Access Right to its Incoming Service Request Queue to Things representing service personnel. One of the service persons takes care of the Maintenance Request using its role on behalf of the Service Organization Thing. SO Thing is logged as the performer and Session ID is logged as in other cases too.
46. Example: Defining contracts, tickets and other agreements as Things +++
    If a contract, ticket or other agreement is given a unique identity, it can be stored as a Thing whether it is physical or purely digital. Signature can be supplied through a service Request. Content can be frozen by removing rights to modify content and Access Rights. Usage can be limited i.e. through Service Request logging or Authenticated Identity Service that keeps track on how many times authentication is used. Uniqueness and ownership can be verifiably transferred when tickets are sold.
47. Example: Spotify –type usage with Thing2Data
    Audio and video content can be attached to Things through Extensions. Content can be streamed using Bots defined by Thing Extensions. Streaming can be activated by an appropriate Service Request procedure where Customer transfers payment to the Purse of the Thing. All transactions are logged to both participants.
48. Example: Customs declaration of a Thing +++
    When a Thing is imported, its Customs Extension is updated with requested data. Customs has an application, which reads the Thing ID and transfers the data from the Customs Extension to regular Customs data structure.
49. Example: Using automobile register plates as Thing Identities +++++
    Anyone can create an application that creates Thing Identities that correspond to automobile register plates. In order to make them Public Identities, one needs to add the Inventory URI, where they initially reside. Similar registry can be done by several Inventories, and the user or the Identifier recognizing application needs to

know and supply the Inventory URI where the Identities were created as the first part of the full Identity. Always with this kind of unique but partial Identifiers it would be best to agree on one primary source. In case of automobile license plates, collaboration with the corresponding government office would be natural and these License Plate Things could in a way be extensions of the official registry.

50. Example: Using image recognition & GPS to identify a Thing Identity from a database ++++
    Identifiers can be partial, if the application or Identifier Type Recognizer can supply the missing part of Local or Public Identity. If i.e. a Recognizer has access to a database where all lamp posts are stored with their co-ordinates and any sort of unique identity for each, a Recognizer can use the visual image of the lamp post to recognize it as a lamp post and the GPS coordinate of the photo to identify the correct lamp post from the database and thus retrieve the unique part of the lamp post Identity.
51. Example: Creating a service request queue for a basket so that it fetches required Archetype Things instances home from store
    A Service Request is created so that it includes parallel tasks. Each task asks to create an individual Local Thing derived from user provided URI and Archetype ID or EAN Code. After the list is completed, the Service Request continues to request a delivery and asks a delivery receipt.
52. Example: Creating a list of Things to be ordered for a construction site from a CAD/CAM model and routing the Things according to CAM as they arrive ++
    CAD/CAM program creates Local Identities for an Inventory using user URI for planned parts and descriptions for each Thing using a suitable Extension. The said Things are not made yet, but they contain their descriptions and a Service Request that is posted to potential suppliers including an offer-order handshake and delivery routine. Only the accepted offer is delivered. Supplier delivers and tags the physical Thing with corresponding Identifier. When receiving Things to the construction site, ordered Things are identified and routed to the destinations that CAD/CAM system provides.
53. Example: Orderly shutdown and evacuation of Things on alarm notification
    A Service Request named Alarm is created for Thing named Alarm Button. Activating Alarm causes it to be posted on the incoming Service Request Queue of all Things that need to know or react to the Alarm. Bots or IoT-devices poll the queue's of their corresponding Things and cause orderly shutdown or evacuation procedures. Monitoring of the completion of Alarm provides additional safeguard.
54. Example: Transport of dangerous things like dangerous chemicals or explosives +
    A Thing, which is considered hazardous, can contain a Bot, which monitors the Extension where data concerning the status of the Thing is maintained. If the Bot notices anything out of the ordinary, it activates a suitable alarm Service Request.
55. Example: Using Thing2Data as a registry with officially certified properties +++
    It is not necessary to grant all Access Rights to the Owner of a Thing. Some Access Rights can instead be held by officials. An example is the license plate registry, where car ownership information and car technical information is commonly maintained by a government office.