

PragmaDev Project:ETSI M2M

By : Axel JEANNE

31/07/14

real time development tools

Table of Contents

1 -Introduction.....	4
2 -ETSI M2M System.....	6
2.1 - Top level system.....	6
2.2 - Device.....	10
2.2.1 Block diagram.....	10
2.2.2 Device Application (DA).....	11
2.2.2.1 DA - diagram.....	11
2.2.2.1.1 Initialization.....	11
2.2.2.1.2 Registration.....	12
2.2.2.1.3 Waiting subscription.....	13
2.2.2.1.4 Normal & long polling wait.....	14
2.2.2.1.5 Asynchronous wait.....	14
2.2.2.1.6 Hybrid wait.....	15
2.2.2.1.7 External inputs.....	15
2.2.2.2 DA - manager.....	16
2.3 - Gateway.....	17
2.3.1 Block diagram.....	17
2.3.2 Service Capability Layer (GSCL).....	18
2.3.2.1 Registration.....	18
2.3.2.2 Create DA.....	19
2.3.2.3 Update Value.....	20
2.3.2.4 Retrieve device value.....	21
2.3.2.5 Create application registration.....	22
2.3.2.6 Delete NA subscription.....	23
2.3.2.7 Delete Device application.....	24
2.4 - Network.....	25
2.4.1 Block diagram.....	25
2.4.2 Service Capability Layer (NSCL).....	26
2.4.2.1 Registration.....	26
2.4.2.2 Create NA.....	27
2.4.2.3 Application ask device value.....	29
2.4.2.4 Update SCL.....	29
2.4.2.5 Notify subscribed application.....	30
2.4.2.6 Delete NA.....	30

2.4.2.7 Delete SCL.....	32
2.4.3 Network Application (NA).....	32
2.4.3.1 NA - process.....	32
2.4.3.1.1 Initialization.....	32
2.4.3.1.2 Waiting NA registration.....	33
2.4.3.1.3 Waiting poll time.....	34
2.4.3.1.4 Waiting notification.....	34
2.4.3.1.5 External inputs.....	35
2.4.3.2 NA - Manager.....	35
3 -Additional Elements.....	37
3.1 - Configuration.....	37
3.1.1 Block diagram.....	37
3.1.2 Process.....	37
3.2 - ASN.1 Library.....	39
3.3 - Abbreviations.....	55
4 -Conclusion.....	57

1 -INTRODUCTION

This project shows how the Real Time Developer Studio (RTDS) can be used to establish a model for the ETSI M2M protocol.

M2M stands for Machine to Machine communication. Today, it is widely used for mobile phones networks. Actually when a new mobile phone registers into a network, all the registration process automatically perform and the user has no need to intervene during this procedure.

Tomorrow, when houses will be fully connected, when the entire household electrical will have a network interface there will be a need for standardized protocol in order to enable communication between many various devices without juggling from a protocol to another. Moreover, this protocol has to be highly scalable so that it can be deployed on houses, cities or even countries.

The goal of ETSI M2M is to use a M2M communication protocol for Iot (Internet of Things). This protocol is able to use a cross platform, language and hardware agnostic procedures in order to run applications that can synchronize or communicate with M2M devices (or common sensors with communication features).

ETSI M2M uses the Restful architecture making it easy to use thanks to words of the Web (POST, GET, UPDATE and DELETE) and have a polling system that is made to save energy (making it ideal for battery reliable devices).

This project

This project recreates a simple system where a house equipped with sensors use Network applications (NA) for various purposes over the ETSI M2M protocol. For convenience purpose, we took the legacy case #2 from the ETSI M2M documentation where a simple device uses a gateway in order to access the Network Service Capability Layer (NSCL).

Initial condition

Here are listed the different elements of the system:

- A non-initialized gateway called m2mPragmaBox
- A non-initialized m2m cloud with the (fictional) address: `http://pragmadev.m2m.com`
- 2 Devices Applications (DA):
 - A smart meter which can provide remotely the consumed electricity.
 - A heat sensor which can give the current temperature of a room.
- 2 Network Applications (NA):
 - A smart meter poller.
 - A fire station application that have access to the smart meter value and the current temperature of the house.

Optimal proceeding

Firstly, the system will boot the GSCL and NSCL. Then the GSCL will attempt to register to the NSCL. When it succeeds, the GSCL is notified and the gateway is ready to run.

During this registration, it is highly probable that NA and DA have sent a registration request, however as the GSCL and NSCL were booting or registering, their registration won't be noticed.

Then on the next registration query according to the “createDevice1st” variable, the DA or the NA will try to register again. If the NA comes first, it will end as a failure because the devices won't exist in the NSCL. If the DA come first, they will correctly register into the GSCL and then be propagated to the NSCL. Once this step is done, the NAs can correctly register to the NSCL.

When every part is registered, the system can run. Two timers moves the system:

One is a DA for the heat sensor. Each time the timer `tNotify` triggers, the device will give its new value which will be propagated to the applications that subscribed to it.

The second is from the NA poller that will periodically poll the smart meter to get its value.

2 -ETSI M2M SYSTEM

2.1 - Top level system

Here we will take the legacy case number #2 of ETSI M2M Framework.

In this case, there are 3 different types of elements.

Device

In this example, devices are sensors with limited network interface (not m2m sensors).

They have the ability to push notifications or to respond their values when asked.

Gateway

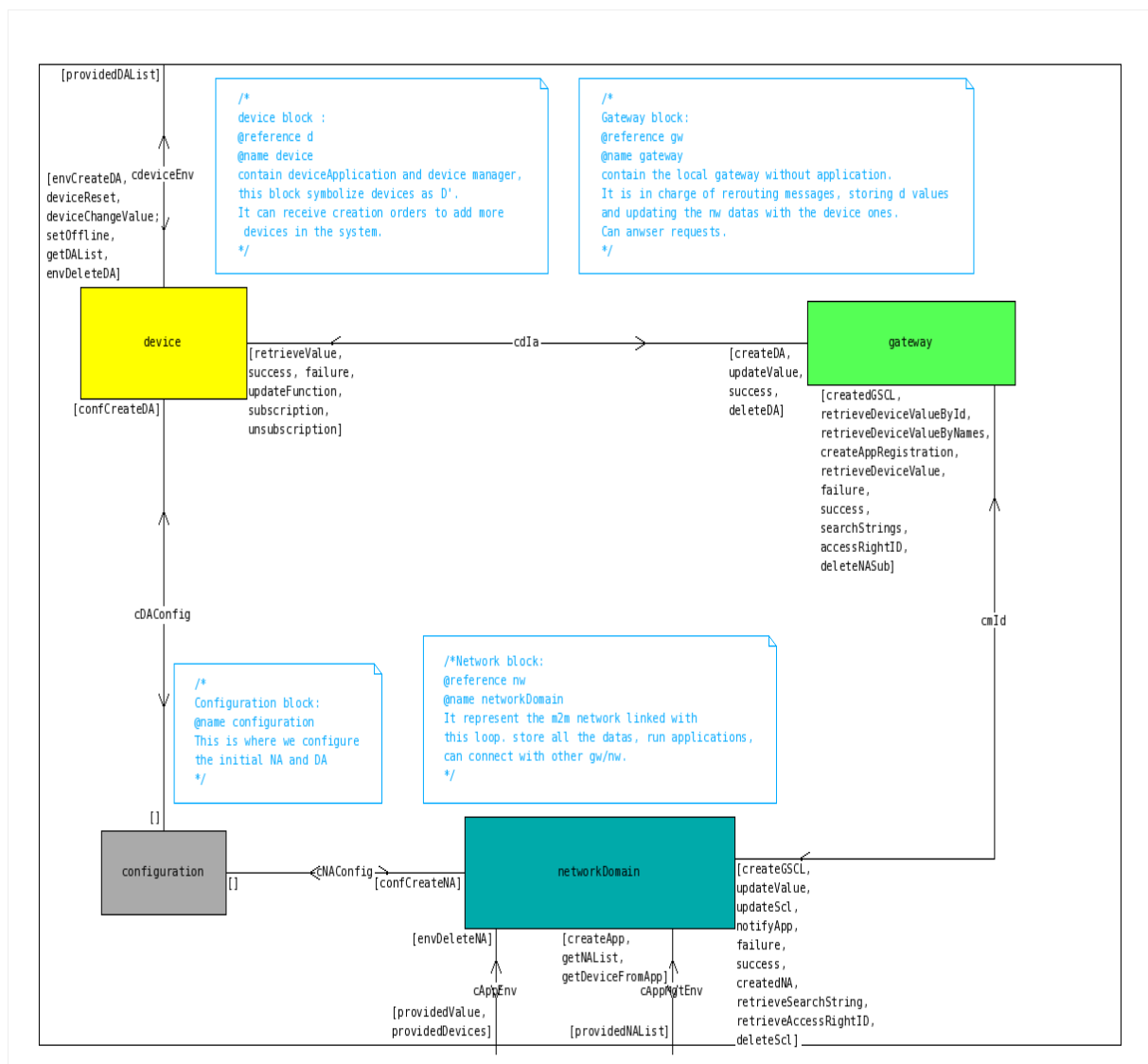
They link the devices with the M2M cloud network, they allow the devices to register, create ressources and communicate using the M2M protocol.

Network

It contains the essence of ETSI M2M. It contains all the datas required about the m2m system. it dynamically add devices and applications that ask for registration.

Configuration

This block is not part of the protocol, it only allows preparing the system's setting and boot e.g. choosing the number of devices / applications, their parameters, which should be booted first, etc.



Additional notes: About the system

This system is closed, it only provides values when required, so the only channels to the environment are from the device and from the network. It can answer a few basic queries (get the device list, get the value of a device by an application, get the application list etc).

As said above, the system takes the legacy case #2 of the ETSI M2M Framework. This does not include m2m devices which don't need a gateway to start running.

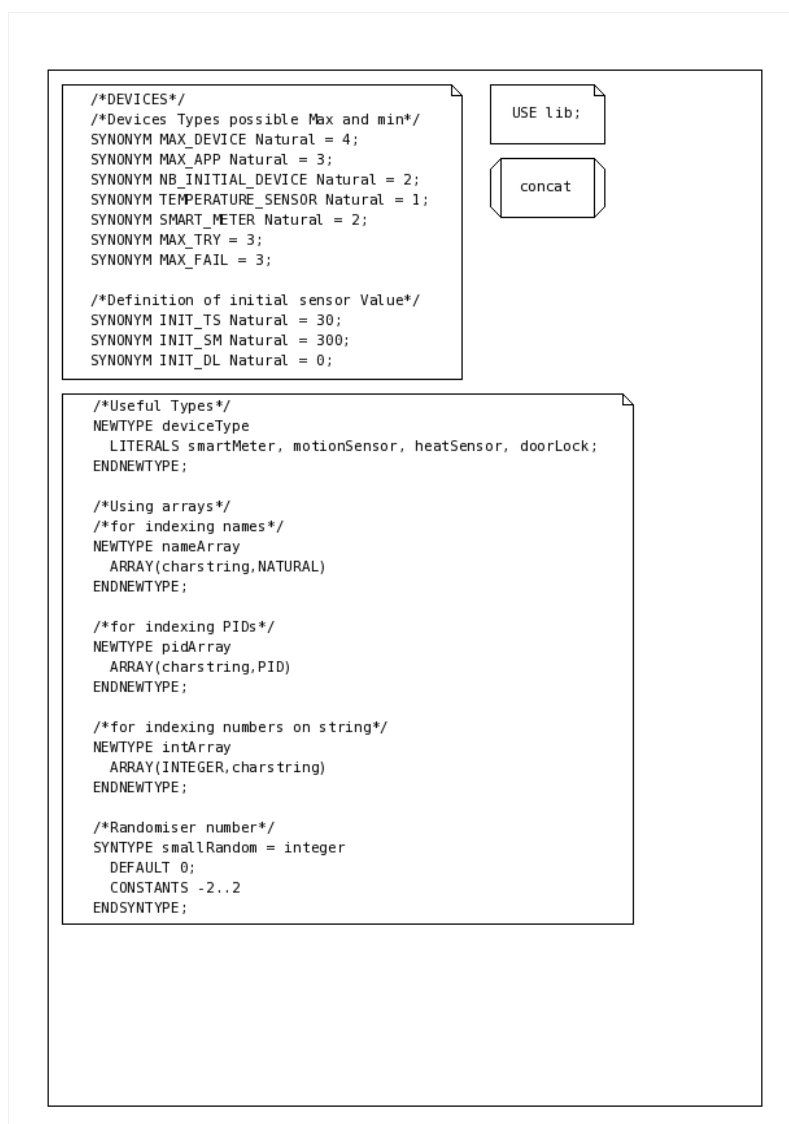
About the features

Some features explained in ETSI M2M are not yet implemented in this system.

Here are declared all the constants, types and procedures that are used among this system.

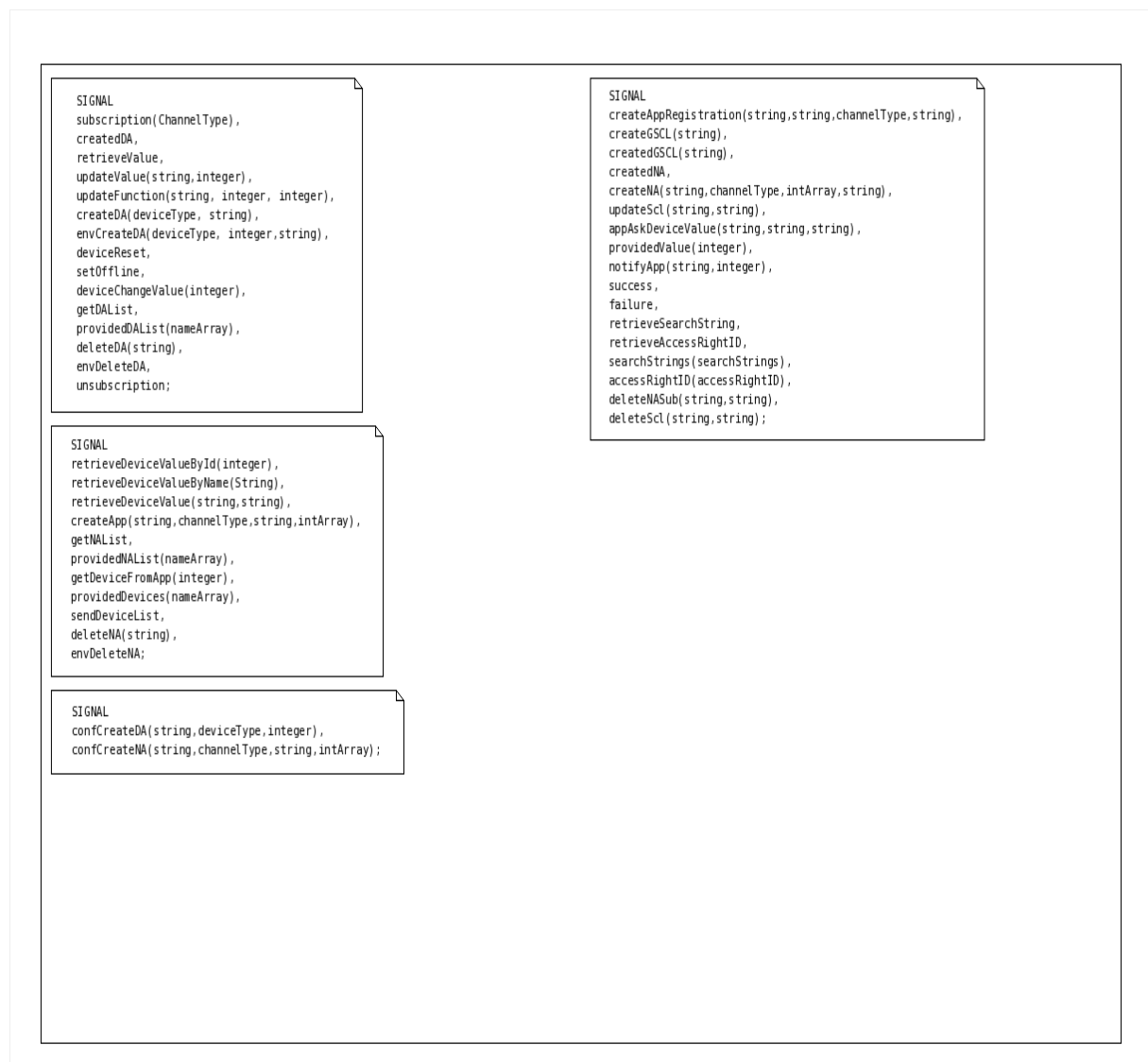
- Constants

- **MAX_DEVICE**: define the maximum number of devices that we can create in this system. This is true for the currently running processes in the system meaning that if we remove and create another device, the system won't forbid it.
- **MAX_APP**: same as **MAX_DEVICE** but for applications.
- **NB_INITIAL_DEVICE**: This parameter is made in order to tell the user how many devices are created by default in this "demo mode". Note: the demo mode is a defined code in the system. Changing this value won't change the number of initial device.
- **MAX_TRY**: This parameter defines how many tries the processes have before aborting the currently running process.
- **MAX_FAIL**: This parameter is the same as **MAX_TRY**. Its only purpose is being changeable by the user to have 2 different constants for abort limit number.
- **Types**
 - **deviceType**: This is the list of handled devices in the system. For each of them, a procedure of **updateValue** is associated. To create new type of device, this list have to be updated and the required behaviour have to be set into **updateValue**. Note: The default behaviour of the procedure **updateValue** is binary: set 0 then 1 each call.
 - **nameArray**: The name array is mostly used for mapping tables into sequences. The ETSI M2M protocol use RESTFUL system which allows to store data under a named array. In order to reproduce this effect with the used ASN.1 script we have, we mapped names with integer in order to find back our stored objects.
 - **pidArray**: **pidArray** is mostly used when a process have to recall of all its neighbour processes and send messages to them. Again the key used is the name and the value associated is a PID.
 - **intArray**: The **intArray** had to be created for tables of devices. We needed a way to loop around all the devices associated to an application. The **intArray** was created to make this procedure possible.
 - **smallRandom**: The **smallRandom** number was made to create more unpredictable evolutions of the values of the sensors. It is currently used in the **heatSensor** for temperature evolution.
- **Library**
 - In this system we only use one library: **lib**. This library currently contain one element: the ASN1 file which contain the mapping of the ETSI M2M ressources used in this system.
- **Procedure**
 - The **concat** procedure was created to fill a compatibility problem between **IA5String** type of ASN1 and **charstring** type from SDL.



Signals

This partition is where all the signals are declared. This allow to define what are the variable type which are provided with messages across communication channels.

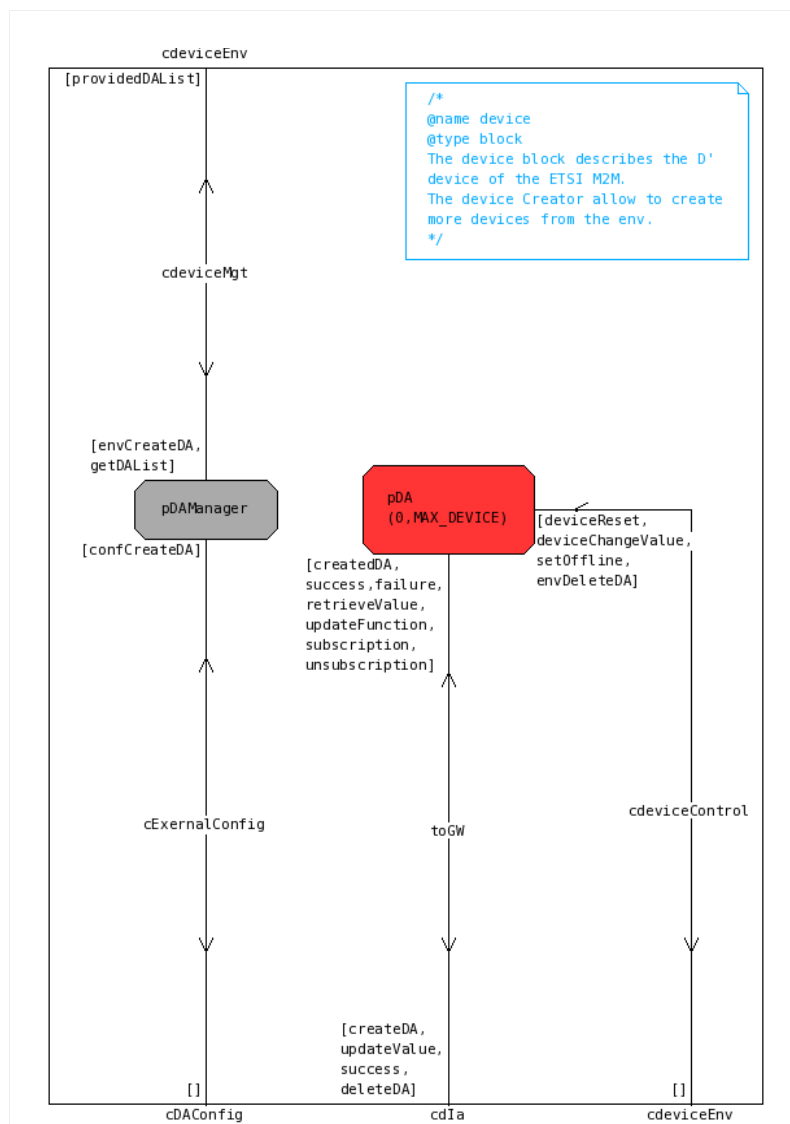


2.2 - Device

2.2.1 Block diagram

The device block regroup the different element of the device D' (described in the ETSI M2M doc) the device is sybolized by the pDA process, which is a basic sensor of any type listed in the topLevelDeclaration.

pDAManager is the process which create new devices. It can also respond to the environment to different queries.



Additional notes

There is no need for communication channel between the pDAManager and pDA, the manager only create and keep a record of the created devices.

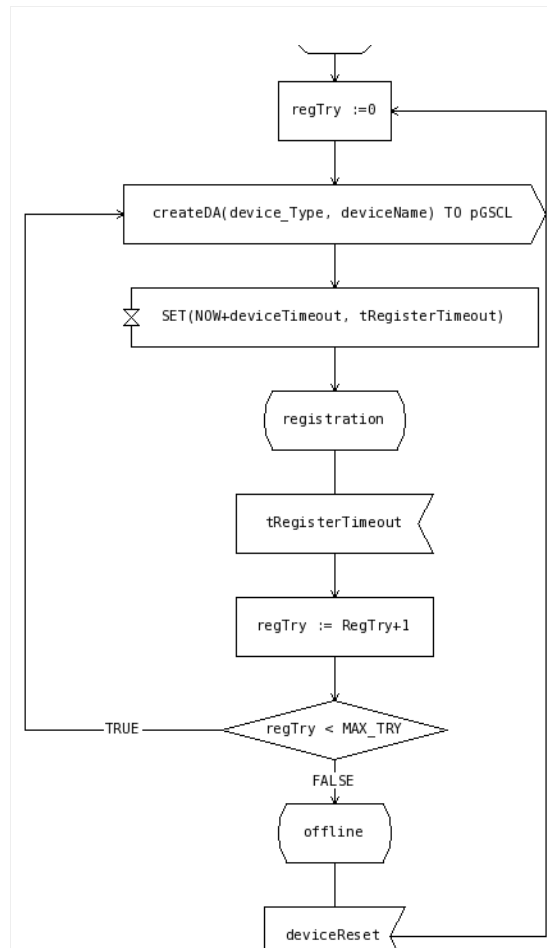
2.2.2 Device Application (DA)

2.2.2.1 DA - diagram

2.2.2.1.1 Initialization

When the device boot it will immediately try to register to to the gateway.

This transition show the pattern that will follow the device when no gateway responds: the registerTimeout will trigger and the regTry counter will increase. If this counter reaches its maximum, the device will go offline and require a manual reset in order to retry the registration procedure.



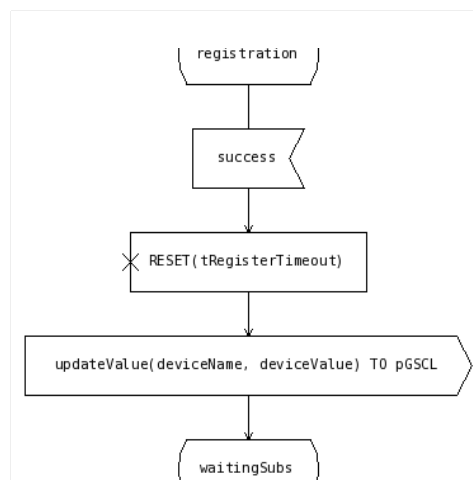
Additional notes

This system is designed for 1 gateway only but the ETSI M2M protocol take into account that multiple gateways can be in range of the device.

The constant `MAX_TRY` is defined in the TopLevel Declarations.

2.2.2.1.2 Registration

When the registration succeed, the registerTimeout is set off and the device can update its first values into the GSCL.



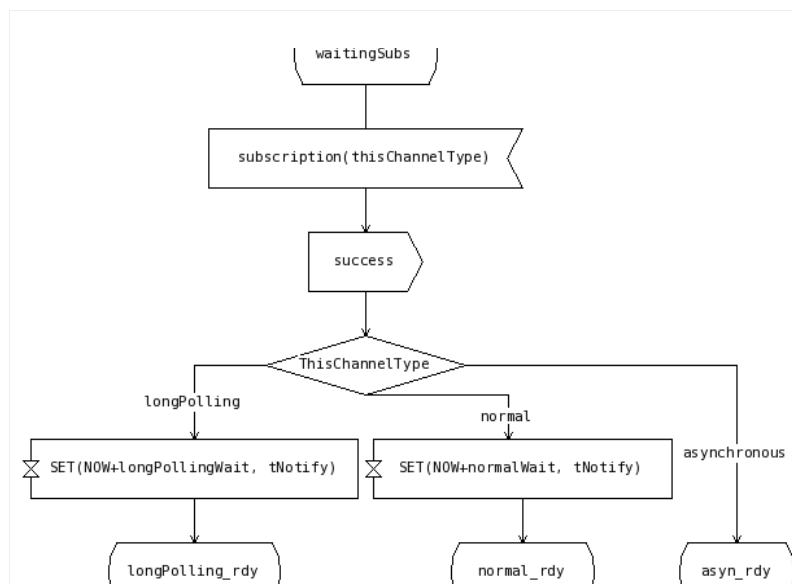
Additional Notes

Once done, the device enters into a waitingSubs state where the sensor is waiting for an application to subscribe in his GSCL <subscription> section.

2.2.2.1.3 Waiting subscription

Once an application have subscribed to the device, it has to give the channelType of communication.

- 1- The normal way: the device will update its value only when it changes from previously stored value (here as we have a determinist system, we create a procedure called "updateValue" which dynamically change the value of the device).
- 2- The asynchronous way: the device will remain passive as long as no polling has been received by the device from the gateway or an application.
- 3- The long polling way: the same as asynchronous way but with a longer response time.



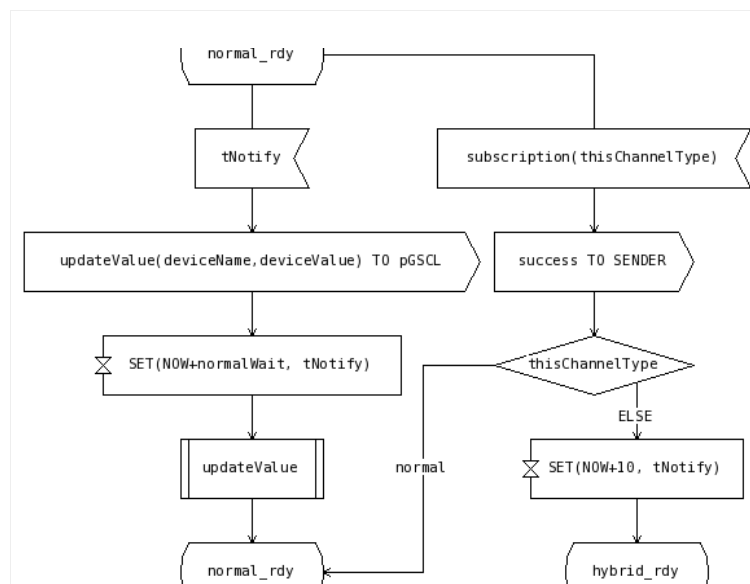
Additional Notes

The type "channelType" is the same as defined in the ETSI M2M documentation (refer to the ASN1 file for more details).

2.2.2.1.4 Normal & long polling wait

When the device enters in the normal state, it remains in a loop in which each step will emit a notification and then change the value of the device (normally it should update the value only when the deviceValue have changed from previous value but the system here is manually stimulated to appear more dynamic).

The device must also be aware of new applications subscribing, so when a new subscription is detected, the device will check if it is of the same type as it is. According to the type it will remain in the state or switch for the hybrid mode.



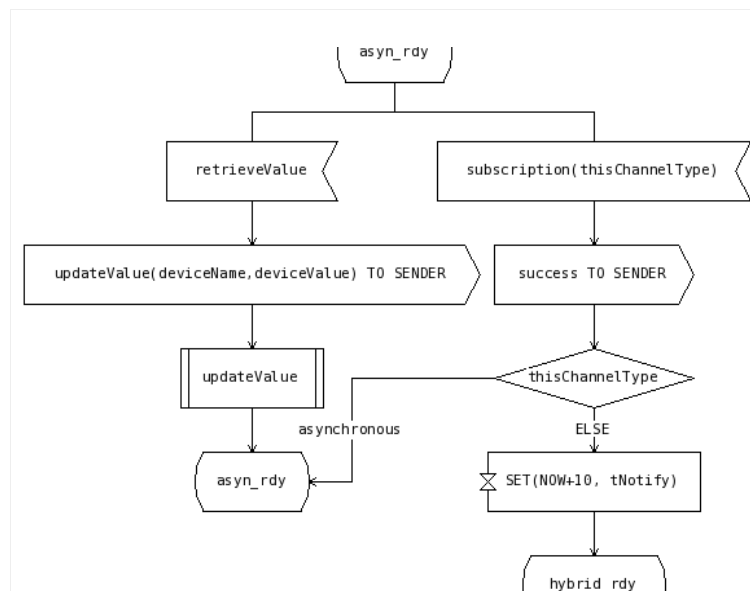
DA/Long polling

The long polling mode is the same as normal but with a longer time period. However we keep it as a separated state because it can handle other feature (for instance going through firewalls etc.)

2.2.2.1.5 Asynchronous wait

In the asynchronous channelType, the device will only respond to a query of an application.

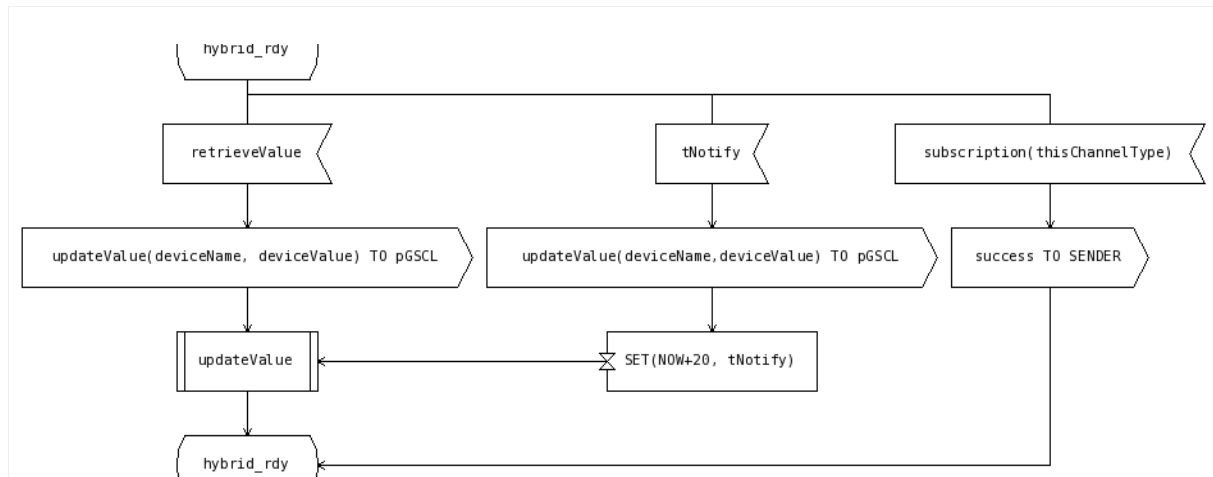
Same as for the normal state, the device will take into account the subscription of new application.



2.2.2.1.6 Hybrid wait

The hybrid mode allows the device to handle multi-application subscriptions which did not shared the same channelType.

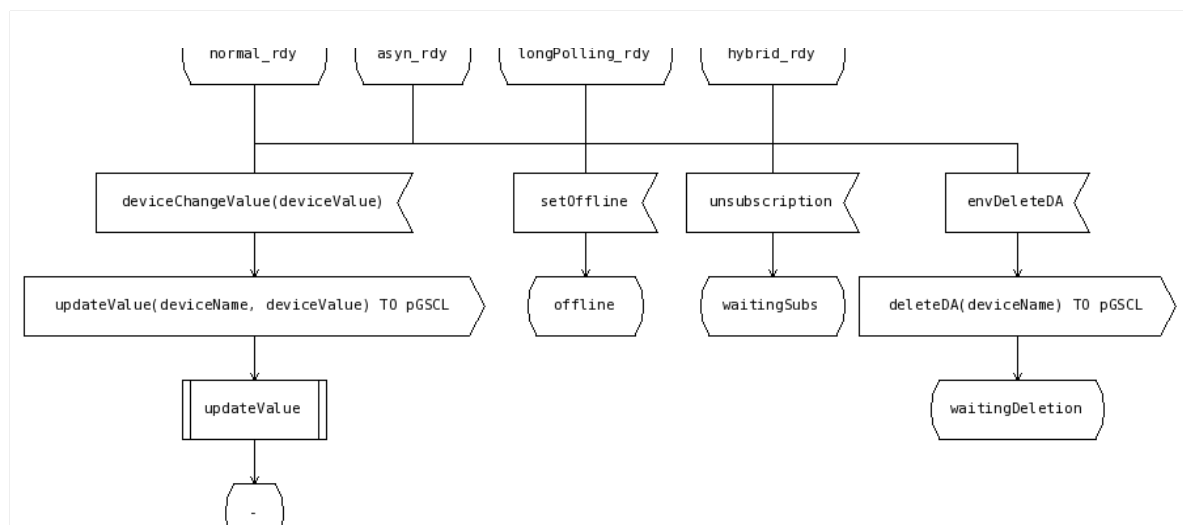
The hybrid mode is the merge between normal_rdy and asyn_rdy. The longPolling_rdy state could not be handled because of the required specific time and features.



2.2.2.1.7 External inputs

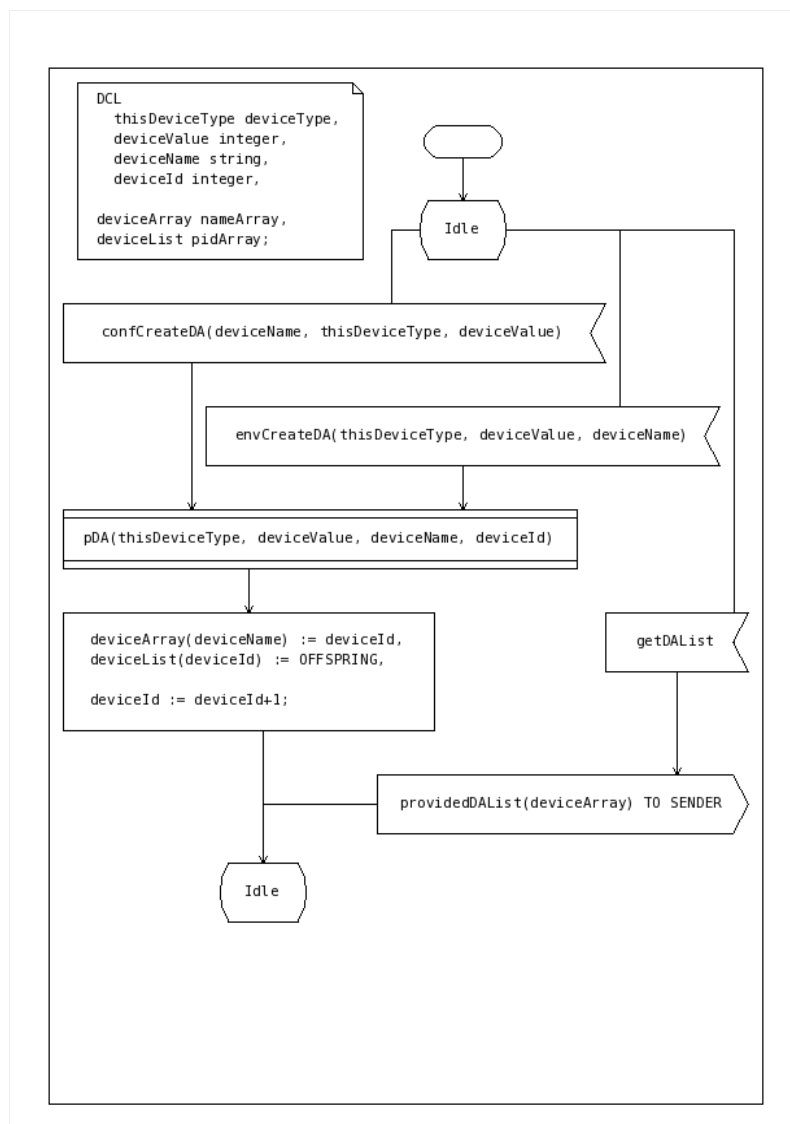
As the device runs, it must be able to receive order from outside, the device can handle multiple queries from the system:

- **DeviceChangeValue** : The value of the device is changed (use with caution as some value can't be negative or null).
- **setOffline** : Set the device offline (but can be set to online with manual reset). Note: this does not allow to bypass the `MAX_DEVICE` limit.
- **unsubscription**: when all subscriptions of the device has been removed, the device come back to the `waitingSubs`
- **envDeleteDA**: try to delete the DA process from the system. The device first notify his suppression to the GSCL until its accepted and then end.



2.2.2.2 DA - manager

The PDAManager handle different kind of request from then environnement/configurator. Basically it is the creator of devices (of the D' type referring to the ETSI M2M documentation). DAM can also respond to a request about the listing of created devices.



Additional notes

Here as an example, we imported the DCL block. It is used to declare all the variables which are used among this process. Declarations are made as <variable Name> <variable Type>. Those variables can be used anytime in this process.

PDAM keep a record of created devices only, this mean that even if the devices have been removed from the GSCL, it will be displayed.

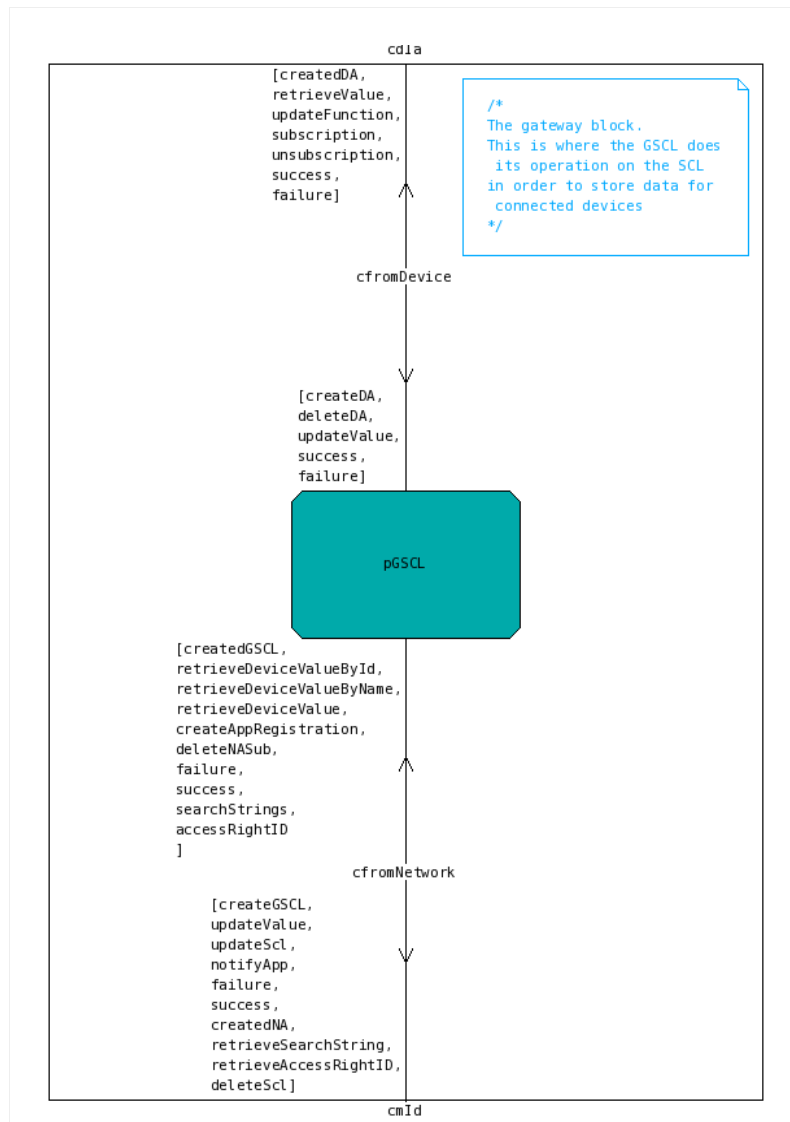
PDAM also keep a track of the adress of each device and store in memory the PID of each created pDA (same remark as above).

2.3 - Gateway

2.3.1 Block diagram

In the gateway block, there is only the pGSCL element because in this example, we only run applications from the network (NA) however the ESTI M2M protocol take into account GA (Gateway Application) that locally run on the gateway.

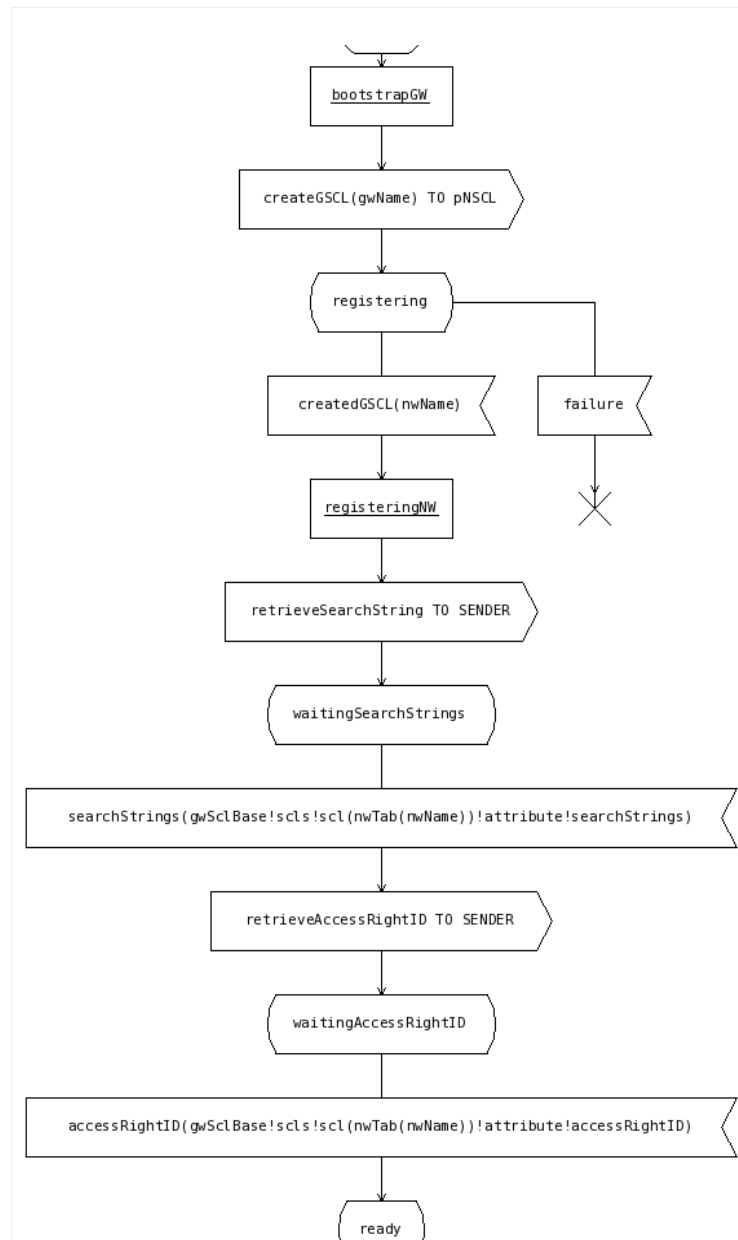
In this model the gateway only have a SCL layer, wich handle most of the request and the communication between the device and the network.



2.3.2 Service Capability Layer (GSCL)

2.3.2.1 Registration

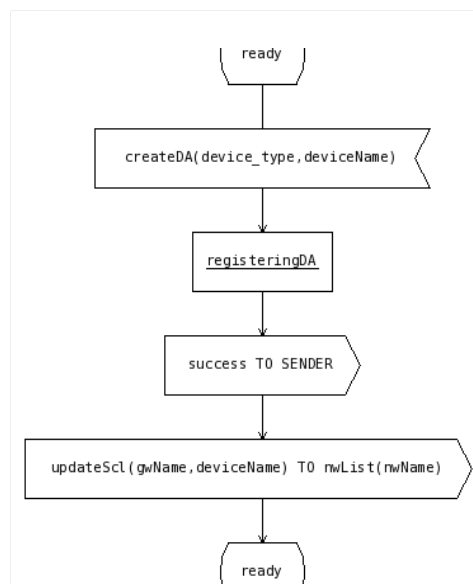
This is the registration procedure of the gateway. Firstly, it will send a creation request to the NSCL. Then when the procedure succeeds, additional informations are provided (searchString, accessRight). Once all the registration has been performed, the GSCL and the NSCL are synchronized and ready to exchange data.



2.3.2.2 Create DA

The **ready** state is the basic state of the gateway. In **ready** the gateway will listen any incoming signal and will start the operation accordingly.

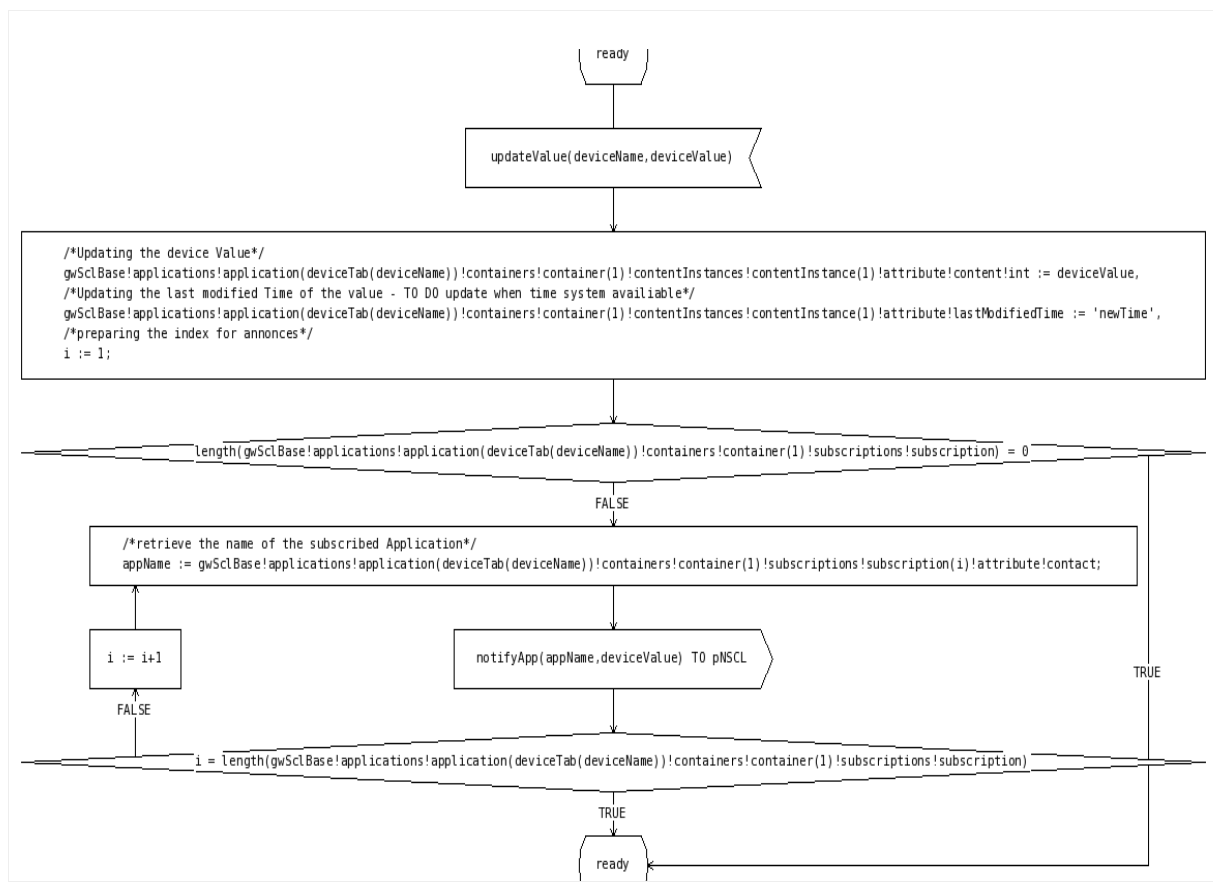
Here the signal is the request of the creation of a new device. The gateway create the local image of device which requested creation. Once done it sends back a success message. And as a consequence of being a remote image, the newly added device must be added into the remote image in the NSCL.



2.3.2.3 Update Value

When a device have to notify because of a new value arrived, it has to send a message to the applications that subscribed in the normal channelType. In order to do this, it sends an `updateValue` signal.

All the following process is made in order to identify every subscribed applications. Once identified, the gateway sends a message with the useful information so that the NSCL can find the application it have to notify.

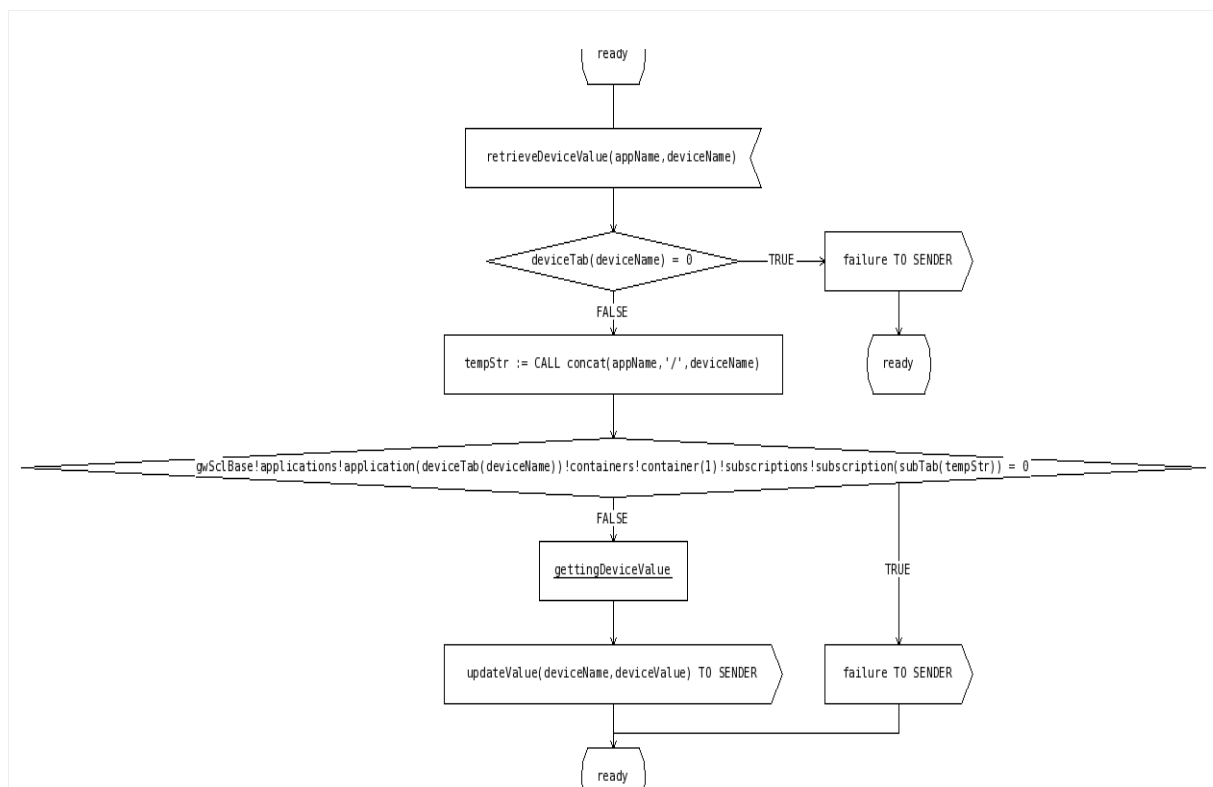


Additional note

It would be eventually possible to modify this sequence in order to add the notification to local applications (GA). The process would be both simpler and shorter (with the condition of having gateway applications already running).

2.3.2.4 Retrieve device value

When an application subscribed to a device has set its `channelType` to asynchronous, the device only respond to external query request. This request is handled when the GSCL receive the `retrieveDeviceValue` signal. When this happend, the gateway asks the corresponding device for its value and then give back the result to the application that asked for it.

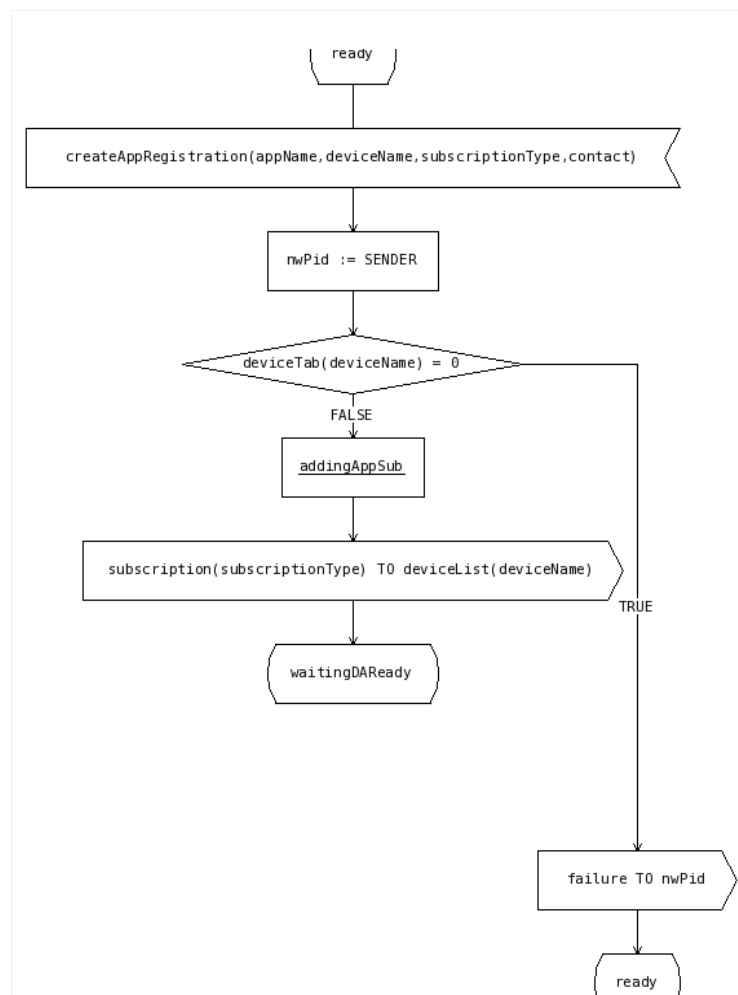


2.3.2.5 Create application registration

When a new application registers into the m2m network, it provides the devices it wants to be connected to.

Once the NSCL registered the application, it asks the gateway a subscription to the indicated devices. If all the procedure runs correctly, it ends with a success message, otherwise failure.

The device is also notified that a subscription has occurred in order to provide the `subscriptionType` and start running.



Additional Notes

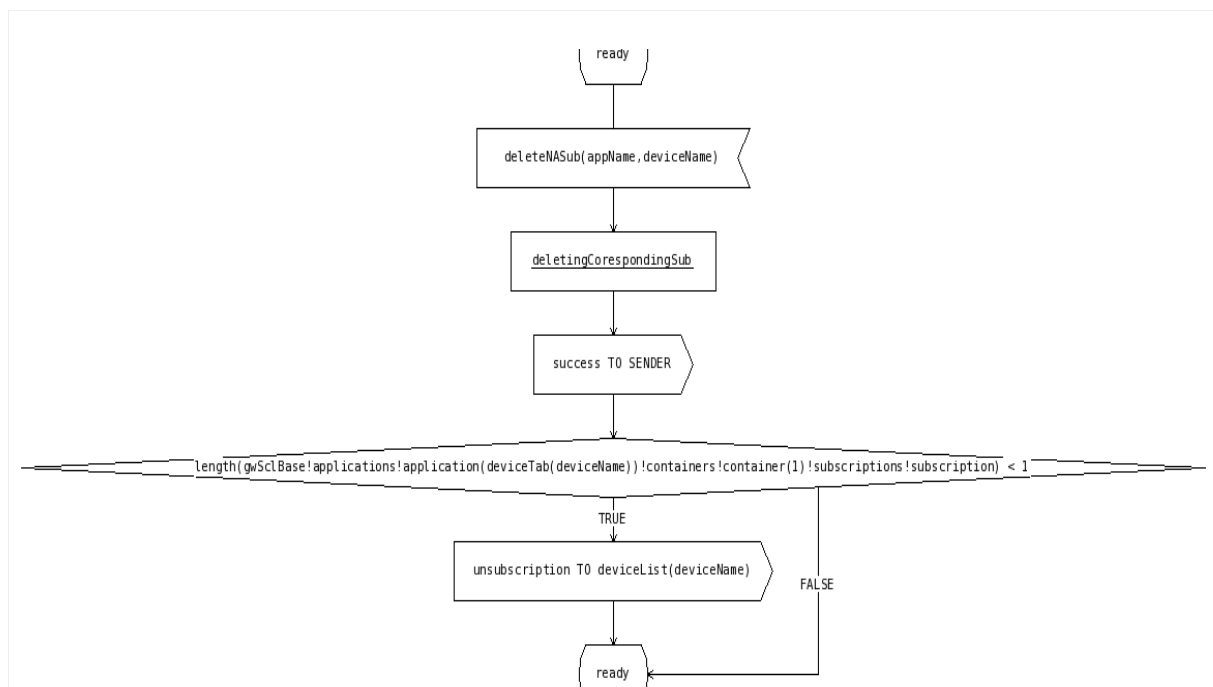
The subscription procedure is mapped on an array which key are stored with the format: "application/deviceName". This is important to note in order to find back the subscription of the device/the application

2.3.2.6 Delete NA subscription

When an application is deleted, every part of the m2m network have to be notified, this is a consequence of the SCL been a mirror image from each other.

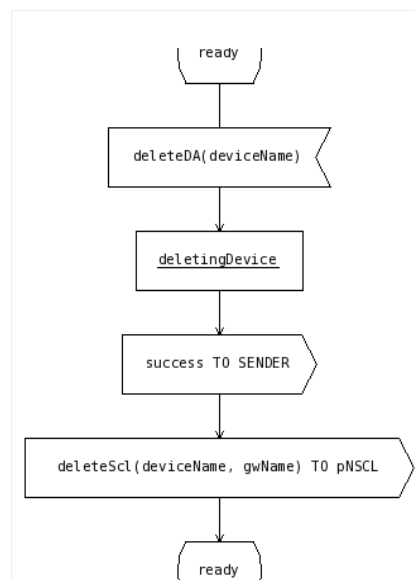
When an application is deleted, before removing it from the m2m network, all the gateways it has subscription to must remove this entry.

This is what happens when the deleteNASub message is received: all the subscriptions are removed, so the application is removed from this local gateway.



2.3.2.7 Delete Device application

When a device wants to shutdown or is going in the "disconnected" status, it first emit (when it can) a `deleteDA` message. When this message is received, then process of deletion of the device start removing all entries from this device in the gateway.



Additional notes

When the device is removed, the subscriptions are not changed.

The testing pattern is set as follows: when an application asks for a device that is not known by the gateway, the response will be a failure. This allows that even if a device goes offline for a short time period, the subscription remains. So there is no need to reconfigure the remote application.

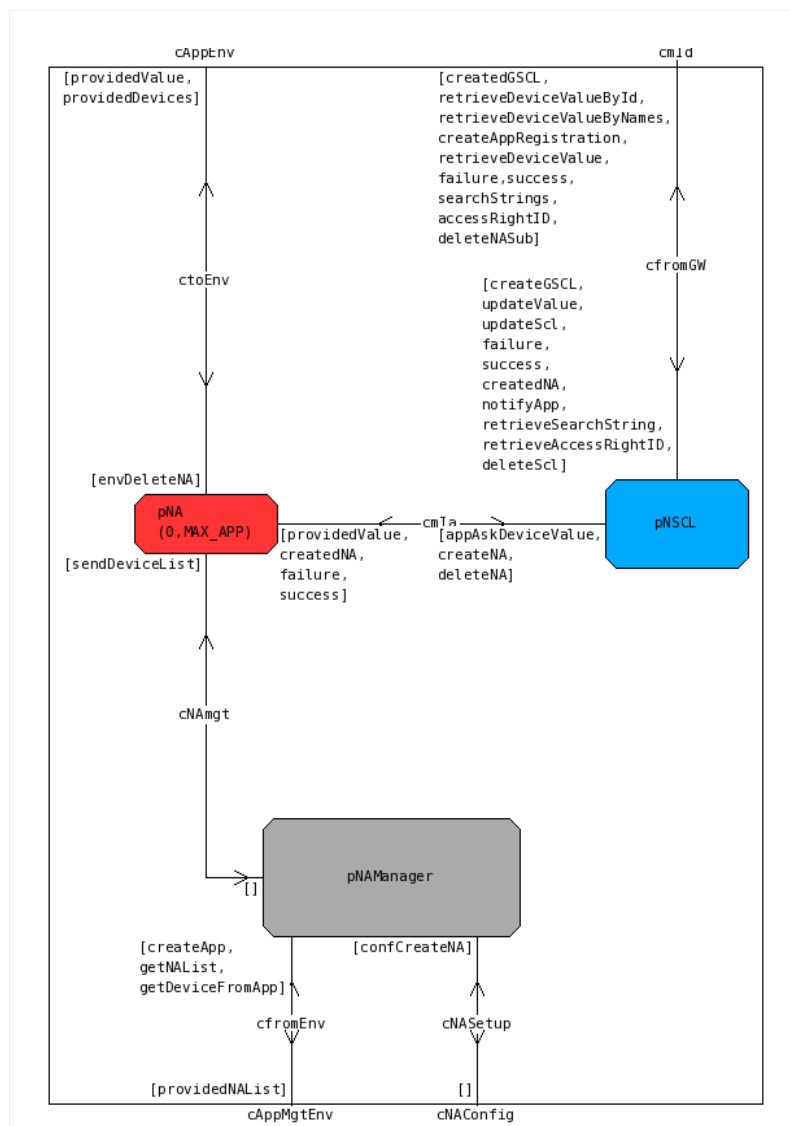
2.4 - Network

2.4.1 Block diagram

The m2m network is the core of the technology.

The NSCL is the biggest part of the system. It contains a mirror image of all the network and elements attached to it.

In this example we chose the application as only network ones (NA).



Additional notes:

This example explains only a part of the ETSI M2M protocol.

It voluntary conceal some aspects of the ETSI M2M for visibility and understability purposes (e.g. AccessRights, Discovery,etc).

2.4.2 Service Capability Layer (NSCL)

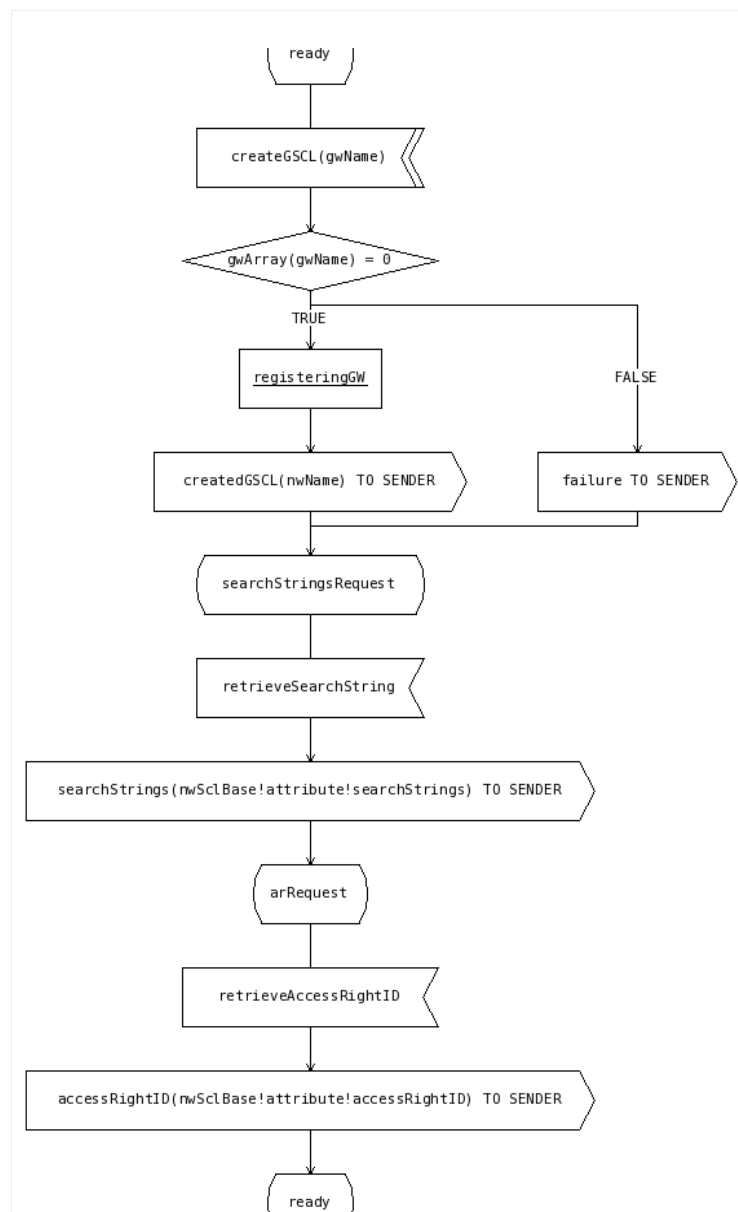
2.4.2.1 Registration

ready

The ready state is the basic state of the NSCL. While in this mode, every signal will be treated and processed.

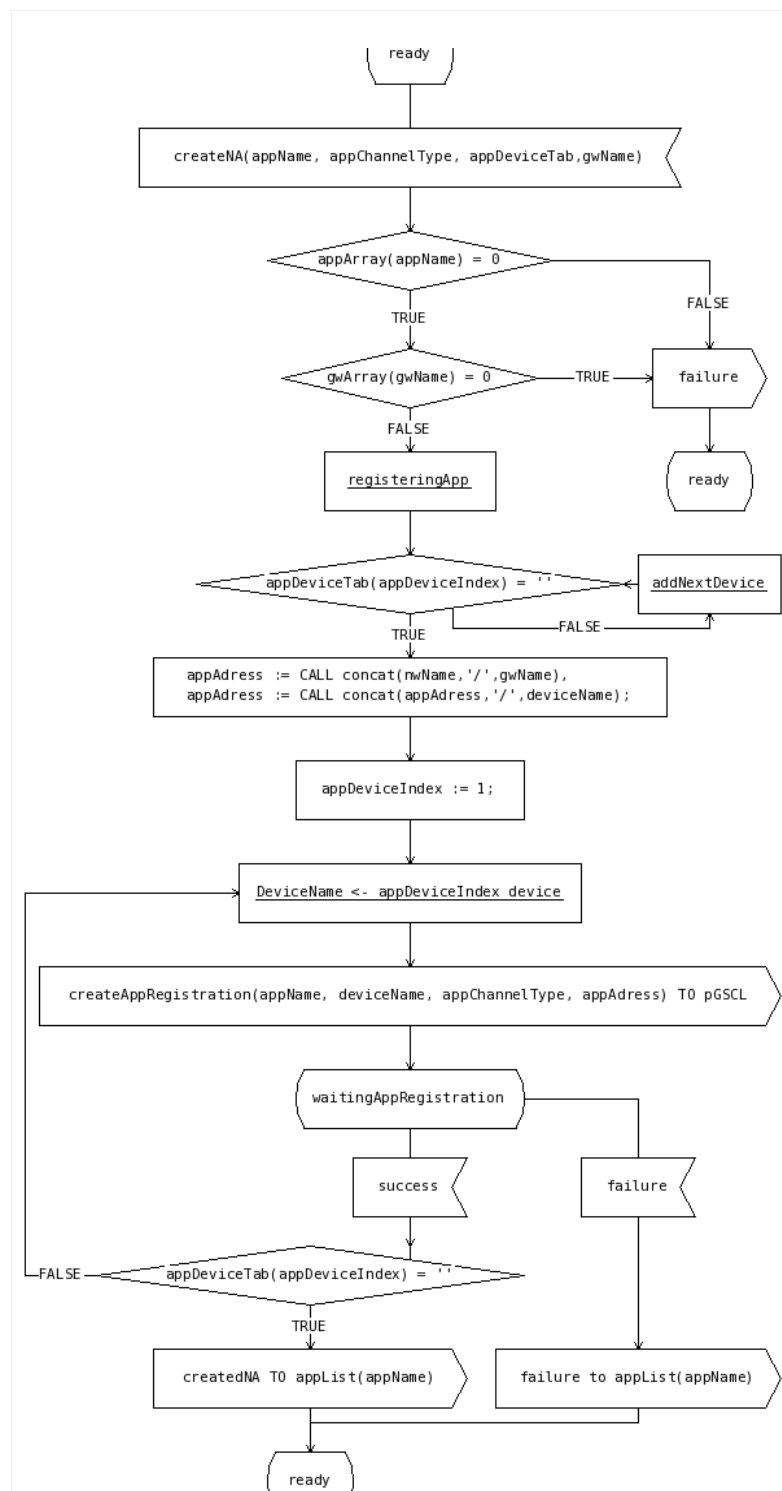
CreateGSCL

When the network receives a creation request from a gateway, it will start registering it in his SCL. Then, it will ask for the searchString and accessRight attribute of it to complete the mirror image it has. Once every parameter has been exchanged, the gateway and network have their corresponding mirror image and are ready to communicate.



2.4.2.2 Create NA

When an application is booted, it asks the NSCL for registration by sending a createNA message. Multiple testing are done in order to check if the application was already registered in the SCL. Then, the gateway the application refers to is checked. If successful the list of devices is verified with a loop. When a device exists, a registration request is sent to the gateway. Then the loop goes on for the next listed device. If all this procedure ran correctly, a success message is sent back to the application otherwise, a failure message.



Additional note:

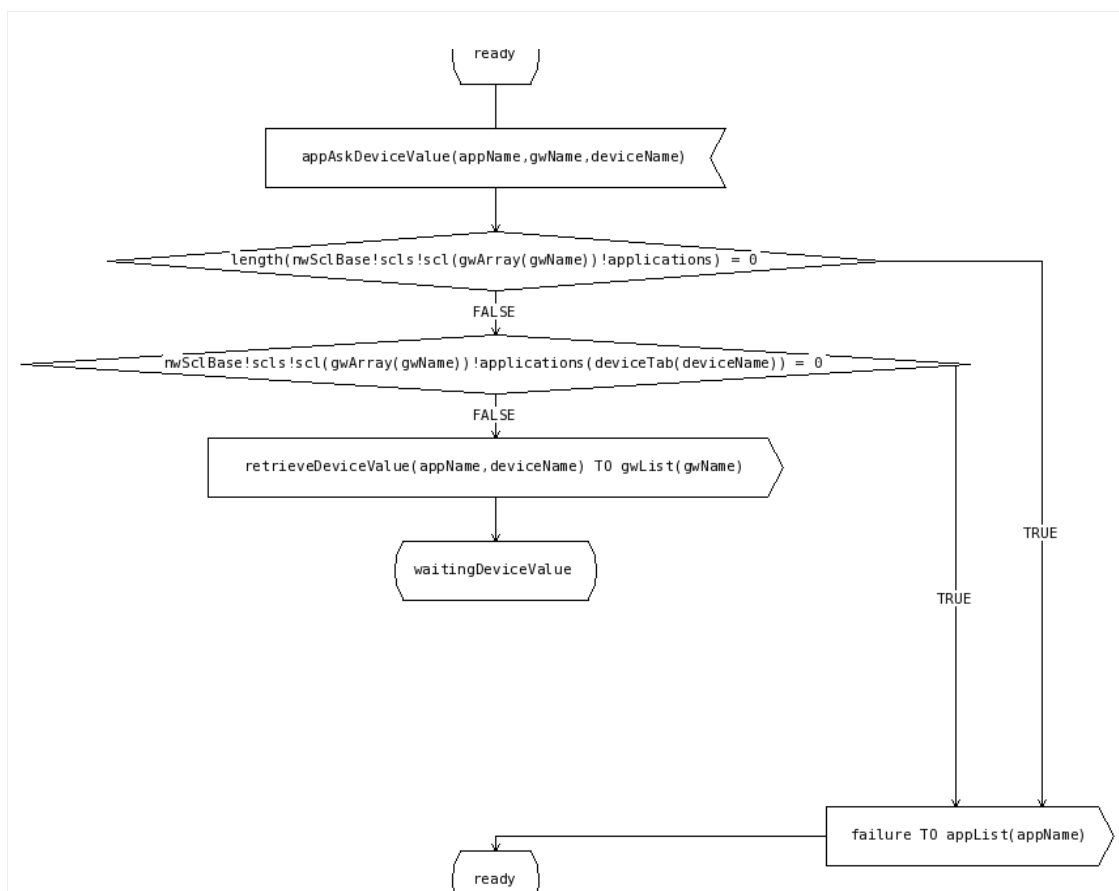
It is important to note that an application that tries to register referring to non-existing device/gateway will result in a failure response message, making the registration process of the application back-off for a time.

An additional feature would be that applications directly register into the NSCL and if none of their devices are available, periodically check for new devices. We didn't implement this mechanism again for simplicity purposes.

2.4.2.3 Application ask device value

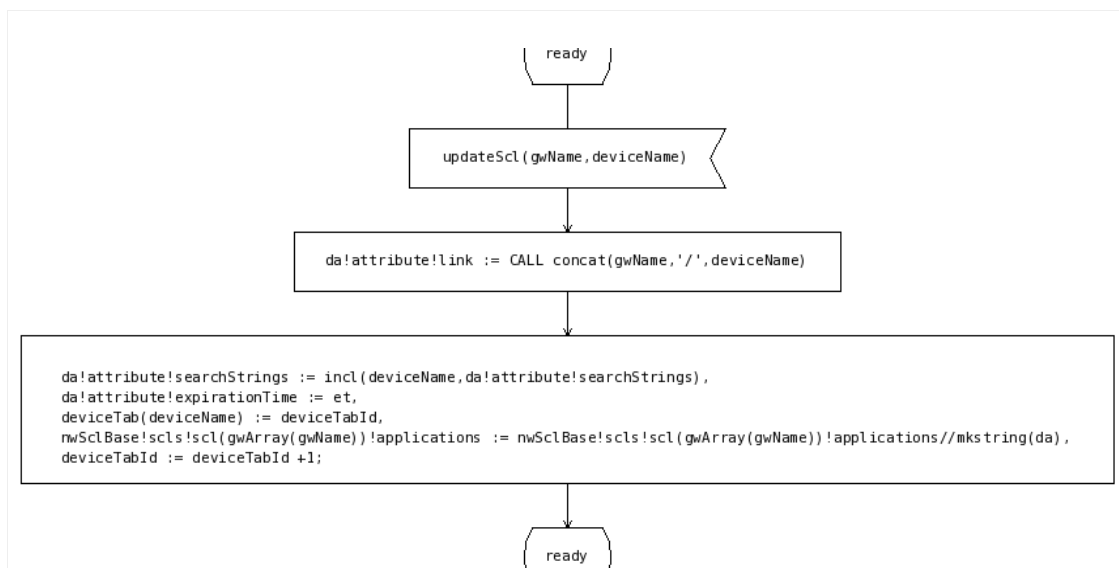
An application using the asynchronous channelType will periodically poll the device it subscribed to. When this happens, it sends an `appAskDeviceValue` message. When the NSCL receives this message, it will poll the devices in its device list through the gateway.

Accordingly to the success or failure of the request, the NSCL transmits the response to the NA.



2.4.2.4 Update SCL

When the gateway has a new element registered it automatically warns the NSCL of its new status. This is part of the "shared mirror image" of the SCLs.



Additional Notes:

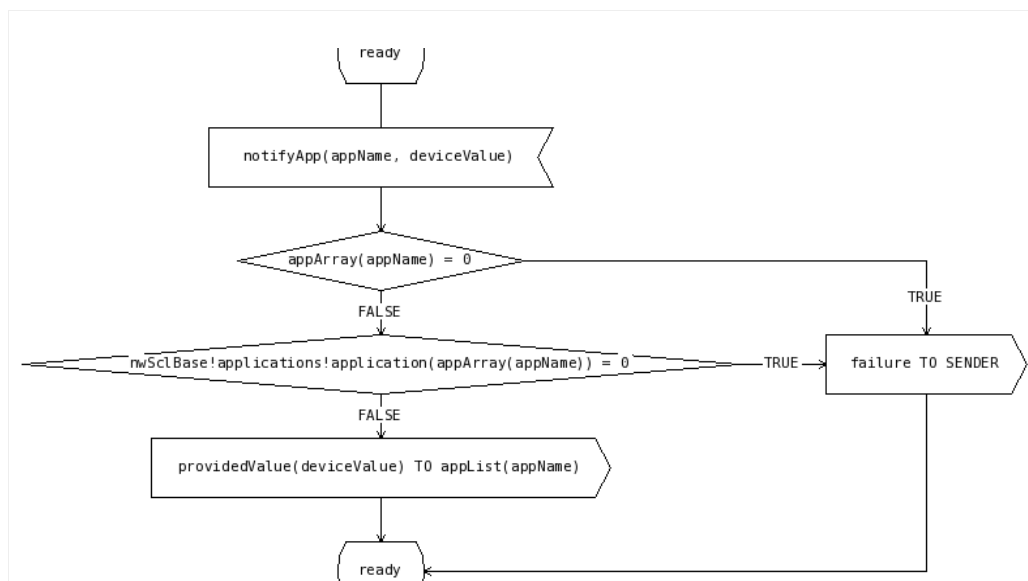
The new scl element in the NSCL is added into sclBase/sclGateway/scls/deviceNameScl.

The affiliated elements are not added under the deviceNameScl. If they are required or accessed, they will be remotely provided through the gateway.

2.4.2.5 Notify subscribed application

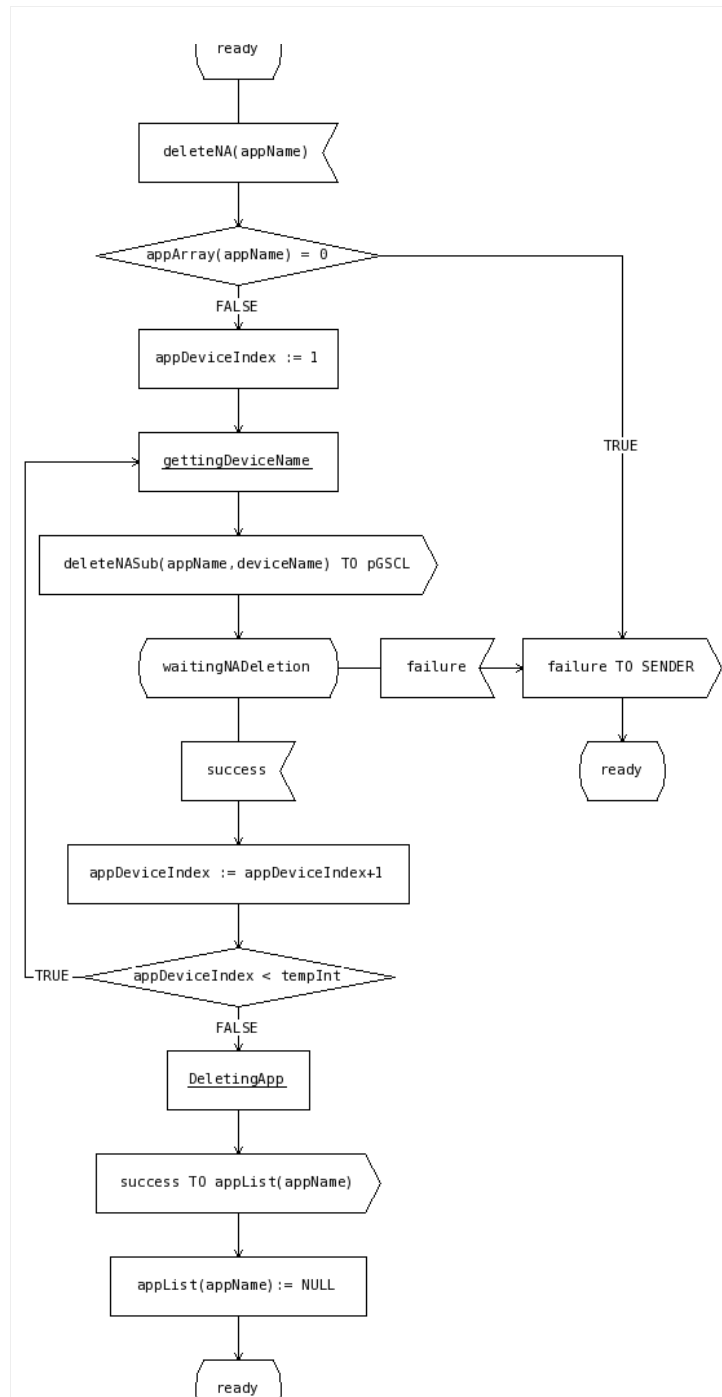
When a device with a subscription notifies an application, it is received by the NSCL with this message.

So when this occurs, the NSCL reroot the value provided to the right NA. If the NA is not found, a failure response is sent back to the sender.



2.4.2.6 Delete NA

When a deletion has been requested by the environment, the NA notifies its deletion to the NSCL. When this message is received, the procedure of subscription deletion starts running. One by one the queries of deletion of subscriptions are sent to the devices through the gateway. Once they are all removed, the application removes itself from the NSCL and it send back a success message to the application.

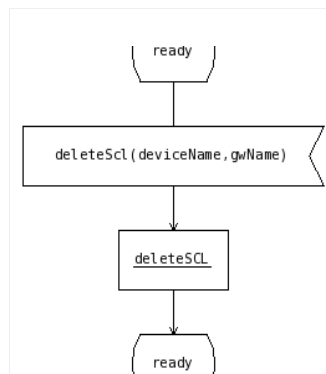


Additional Notes:

The removal process is a complicated algorithm which include loops around sequences of elements in order to find the element to remove of the list and re-create a new list whitout this element in order to simulate a removal. You can have an in depth view of this procedure in the `DeletingApp` code block.

2.4.2.7 Delete SCL

When a device is removed from the gateway, it has to also be removed from the distant mirror image of the gateway in the NSCL. So when this message is received, the NSCL remove the device from the gateway SCL too.



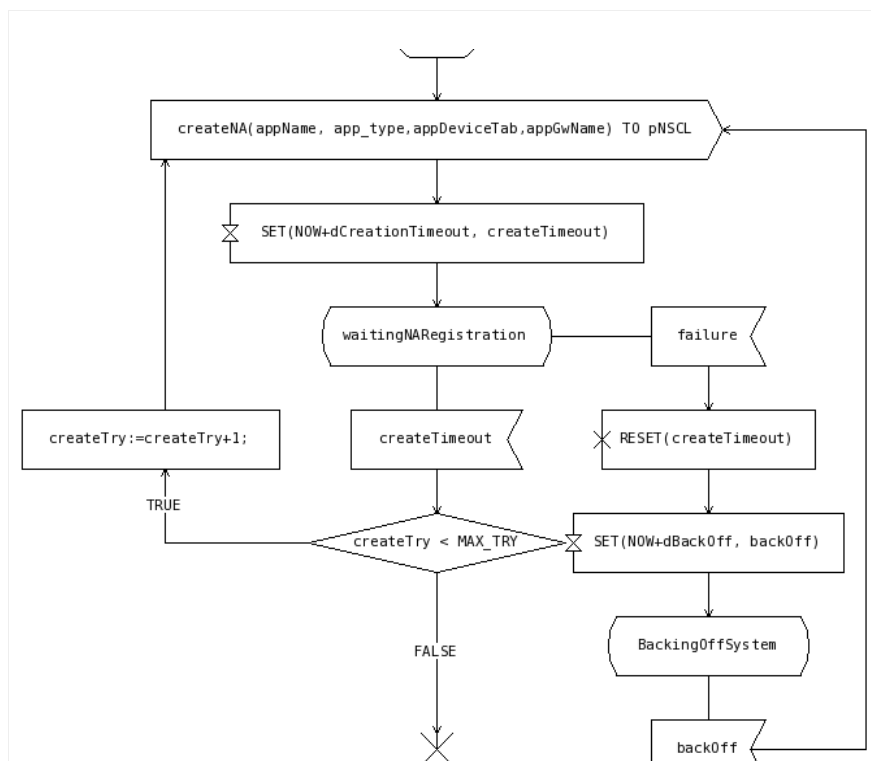
2.4.3 Network Application (NA)

2.4.3.1 NA - process

2.4.3.1.1 Initialization

When a Network Application (NA) is created, it request registration.

If the registration fails, it will back-off for a time and then ask again for registration. The maximum number of authorized registration attempts is set into the main declaration file (here referred as `MAX_TRY`).

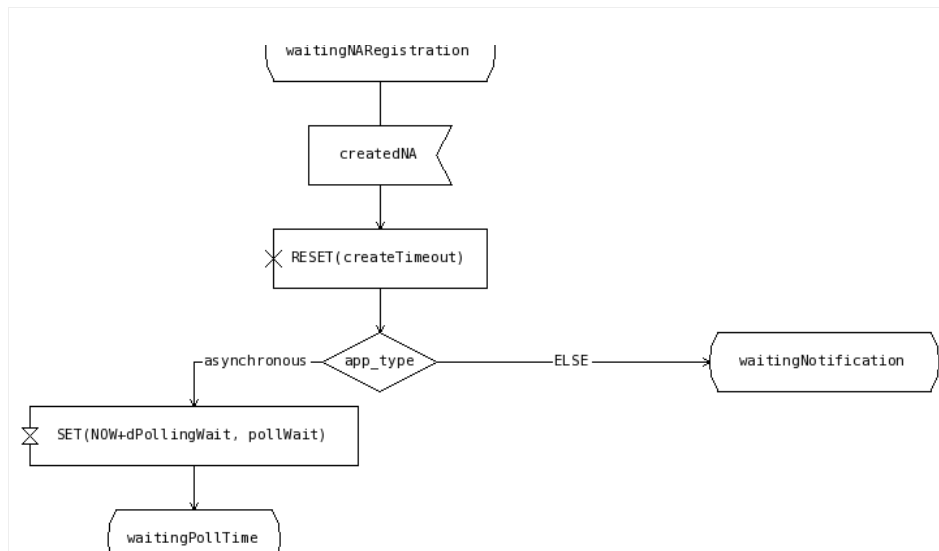


2.4.3.1.2 Waiting NA registration

Once the device is registered in the network, it starts waiting its confirmation of registration from the gateway, meaning that the subscription has been correctly accepted by the device.

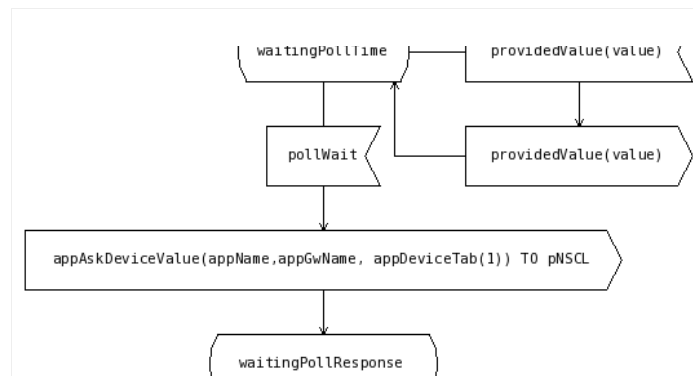
Then the application will start its loop behaviour:

- 1- asynchronous: the application periodically polls the device he subscribed to and then starts waiting for a response.
- 2- normal/longPolling: the application starts sleeping, and is woke up by a device it subscribed to that update its value.



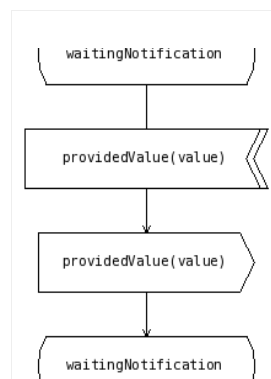
2.4.3.1.3 Waiting poll time

In this state the NA is currently waiting for the timer `pollWait`. This timer triggers a device value request, which is sent to the NSCL.



2.4.3.1.4 Waiting notification

When the device has been set into normal or longPolling mode, the application starts sleeping. It will be woke up by the `providedValue` message that update the value of the device this application subscribed to.

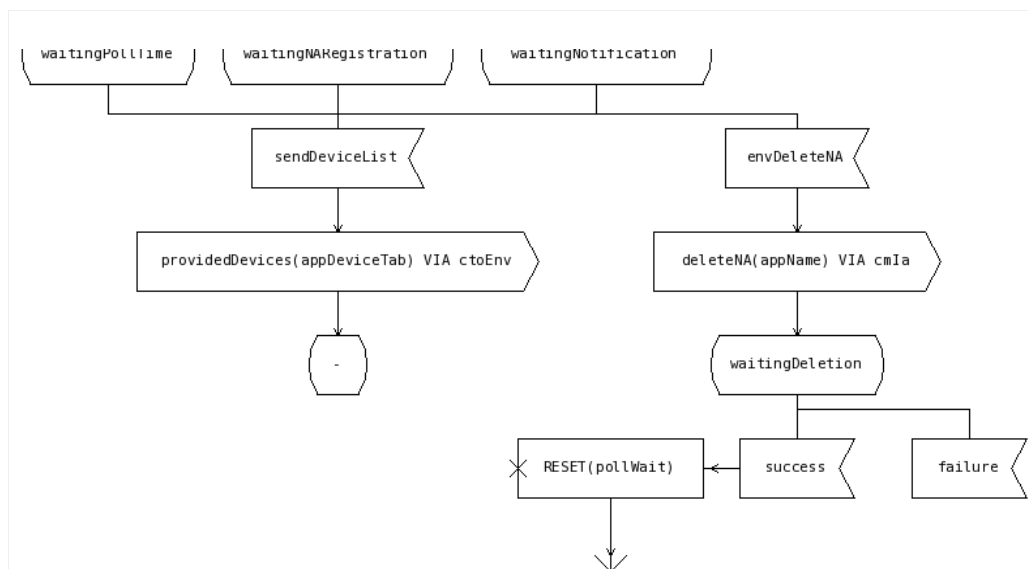


2.4.3.1.5 External inputs

When a NA is in a waiting state, we need it to be watchful from management messages.

In this system we made 2 management messages for NA:

- 1- **SendDevicesList**: This message asks the application for the list of devices it registered to. The response is sent to then environment.
- 2- **deleteNA**: This message starts the deletion procedure of an NA. Then the NA will send a deletion request which propagate to the NSCL and GSCL in order to remove every track of this NA.

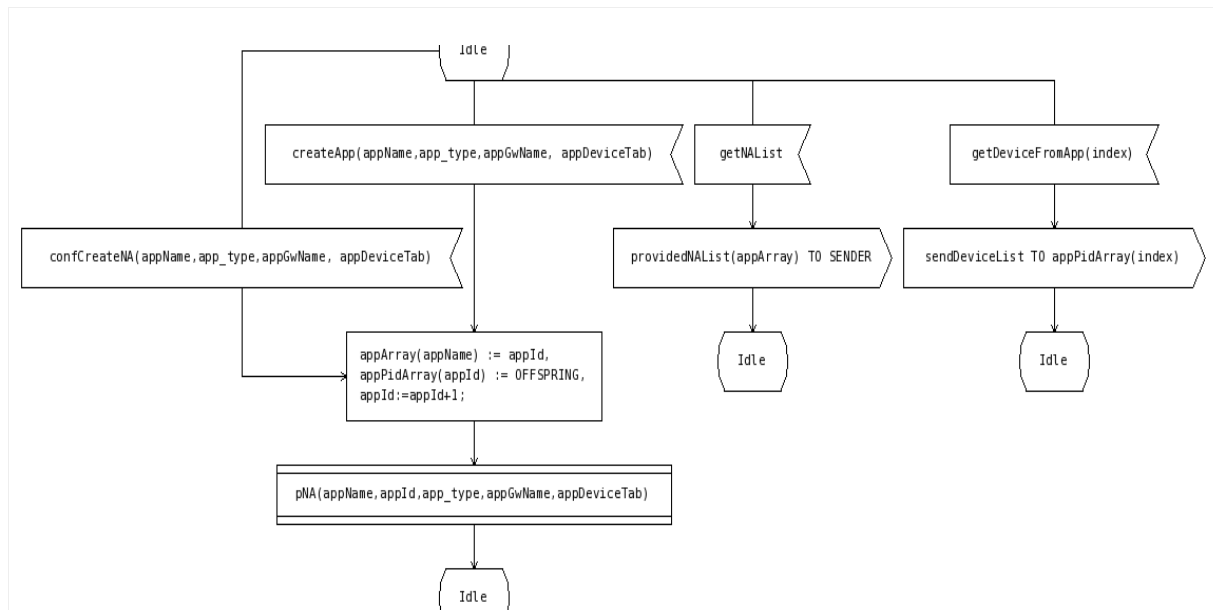


2.4.3.2 NA - Manager

The NManager is the process that handles the creation, configuration and listing of all the NA.

- **Creation**: it can come from the environment (meaning the user manually created a application) or from the configuration (the initial state of the system that we choose).

- Listing: then environnement can request a listing of all the existing application on the m2m server. the NAM will give back the list of the name of all the created application.



Additional notes

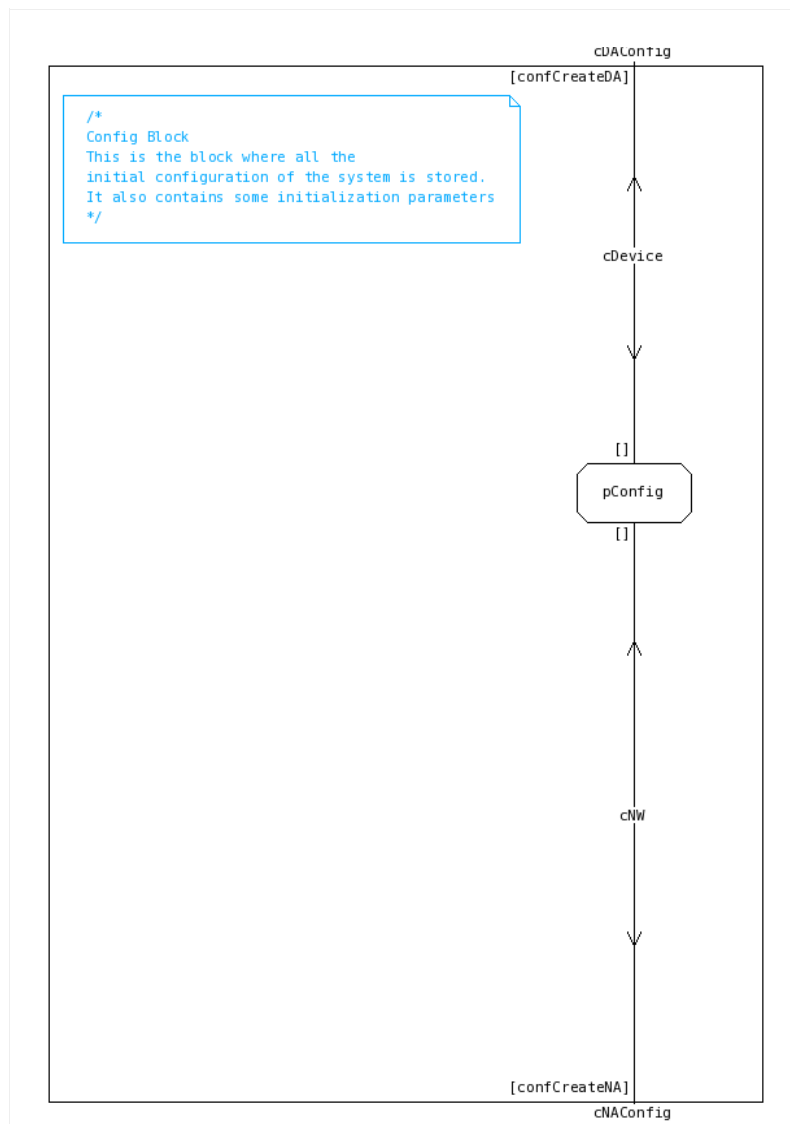
Same remark as the DAM: the NAM will only keep track of the created application.

3 -ADDITIONAL ELEMENTS

3.1 - Configuration

3.1.1 Block diagram

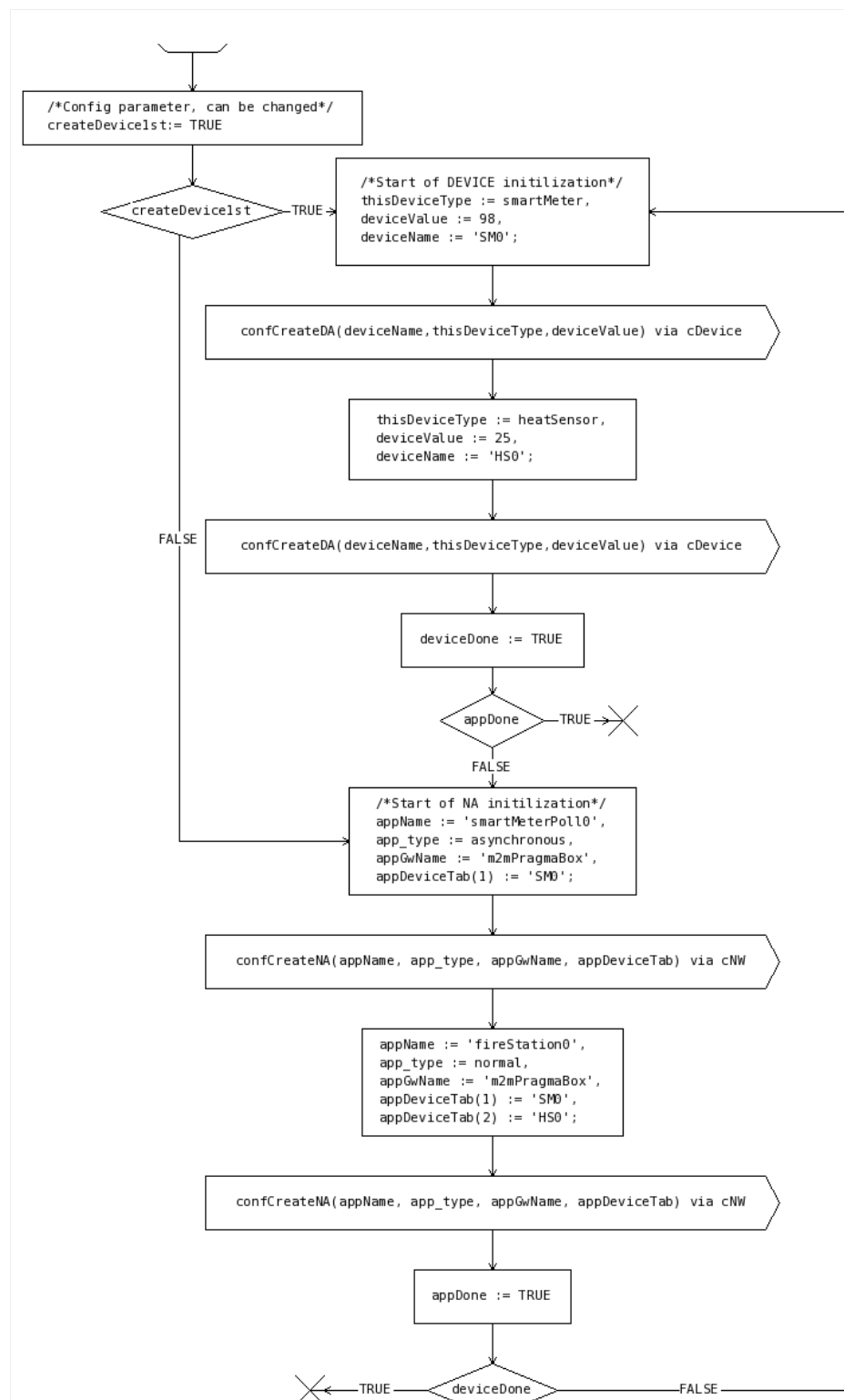
The configuration block was made in order to have a centralized tool for managing the initial conditions of the system.



3.1.2 Process

The configuration of the initial devices and NA is defined here. To customize your system, you have to:

- Choose if the devices are created first or not using the boolean variable `createdDevices1st`.
- In the top section of the diagram (under the Start comment): enter your device type, `deviceValue`, `deviceName` and the the corresponding creation query.
- The maximum number of devices is only conditioned by the `MAX_DEVICE` value.
- Then after the `appDone` condition, (after the Start NA comment). There is the definition then the creation request of NA.



3.2 - ASN.1 Library

-- Creator: RTDS ASN.1 Editor


```
-- Author: Axel J. for PragmaDev
-- Created: Mon May 13 15:24:20 CEST 2014
-- Last Update: Thy June 26 17:33:10 CEST 2014
M2mModule DEFINITIONS ::=
BEGIN

-- imports and exports
--none

-- Special ATTRIBUTES &
CONSTANT

AVERAGESIZE INTEGER ::= 9
HIGHSIZE INTEGER ::= 50
SMALLSIZE INTEGER ::= 2

--TemporalVariable ::= SEQUENCE {
--    year                INTEGER,
--    month                INTEGER (1..12),
--    day                  INTEGER (1..31),
--    hour                 INTEGER (1..24),
--    min                  INTEGER (1..60),
--    sec                  INTEGER (1..60)
--}
String ::= IA5String
TemporalVariable ::= String
Content ::= String
Permission ::= String

--Useful Sequences/sets
SetsOfFilterCriteria ::= SET (SIZE(0..9)) OF String
SetsOfSearchStrings ::= SET(SIZE(0..9)) OF String
SetsOfDiscovery ::= SET(SIZE(0..9)) OF String
ListOfM2mPoc ::= SET (SIZE(1..9)) OF M2mPoc
ContentData ::= CHOICE {string [0] String,int[1] INTEGER,bool [2] BOOLEAN, dt [3]
DeviceType}

SeqOfScl ::=SEQUENCE(SIZE(0..9)) OF Scl
SeqOfApplication ::=SEQUENCE(SIZE(0..9)) OF Application
```

```

SeqOfApplicationAnnc ::=SEQUENCE(SIZE(0..9)) OF      ApplicationAnnc
SeqOfAccessRight      ::=SEQUENCE(SIZE(0..9)) OF      AccessRight
SeqOfAccessRightAnnc  ::=SEQUENCE(SIZE(0..9)) OF      AccessRightAnnc
SeqOfContainer         ::=SEQUENCE(SIZE(0..9)) OF      Container
SeqOfContainerAnnc     ::=SEQUENCE(SIZE(0..9)) OF      ContainerAnnc
SeqOfLocContainer      ::=SEQUENCE(SIZE(0..9)) OF      LocationContainer
SeqOfLocContainerAnnc  ::=SEQUENCE(SIZE(0..9)) OF      LocationContainerAnnc
SeqOfContentInstance   ::=SEQUENCE(SIZE(0..9)) OF      ContentInstance
SeqOfGroup             ::=SEQUENCE(SIZE(0..9)) OF      Group
SeqOfGroupAnnc         ::=SEQUENCE(SIZE(0..9)) OF      GroupAnnc
SeqOfString            ::=SEQUENCE(SIZE(0..9)) OF      String
SeqOfSubscription      ::=SEQUENCE(SIZE(0..9)) OF      Subscription
SeqOfM2mPoc            ::=SEQUENCE(SIZE(0..9)) OF      M2mPoc
SeqOfMgmtObj           ::=SEQUENCE(SIZE(0..9)) OF      MgmtObj
SeqOfMgmtCmd           ::=SEQUENCE(SIZE(0..9)) OF      MgmtCmd
SeqOfParameters        ::=SEQUENCE(SIZE(0..9)) OF      Parameters
SeqOfLink              ::=SEQUENCE(SIZE(0..9)) OF      Link
SeqOfExecInstance      ::=SEQUENCE(SIZE(0..9)) OF      ExecInstance
SeqOfAttachedDevice    ::=SEQUENCE(SIZE(0..9)) OF      AttachedDevice
SeqOfNotificationChannel ::=SEQUENCE(SIZE(0..9)) OF      NotificationChannel
SeqOfSclAnnc           ::=SEQUENCE(SIZE(0..9)) OF      SclAnnc
Members ::=SEQUENCE(SIZE(0..9)) OF String
MgmtProtocolList ::=SEQUENCE(SIZE(0..9)) OF MgmtProtocol
SeqOfPerm              ::=SEQUENCE(SIZE(0..9)) OF      Permission
SeqOfAppAnnc ::=SEQUENCE(SIZE(0..9)) OF ApplicationAnnc

--Enumerated Types ICI TOUS LES TYPES ENUMERES
OnlineStatus ::=ENUMERATED{ online(0), offline(1), notReachable(2)}
ChannelType  ::=ENUMERATED{ longPolling(0), normal(1), asynchronous(2)}
APocHandling ::=ENUMERATED{      deep(0),      shallow(1)}
MgmtProtocol ::=ENUMERATED{ sclRegistration(0), updateSclRegistration(1),
normalRestfulUpdate(2)}
SclType       ::=ENUMERATED{      nscl(0), gscl(1), dscl(2)}
ReferencePoint ::=ENUMERATED{ miaReferencePoint(0), diaReferencePoint(1)}
ExecStatus    ::=ENUMERATED{ initiated(0), started(1), finished(2), cancelled(3),
deleted(4)}

-- type assignments

```

```

-- COMMON ATTRIBUTES
--
AccessRightID ::= String
--Deny loops
--AnnounceTo ::= SEQUENCE OF Scl
AnnounceTo ::= SEQUENCE(SIZE(0..9)) OF String
CreationTime ::= TemporalVariable
ExpirationTime ::= TemporalVariable
FilterCriteria ::= SetsOfFilterCriteria
LastModifiedTime ::= TemporalVariable
Link ::= String
SearchStrings ::= SetsOfSearchStrings

-- Basic ATTRIBUTES by Type
ASclBase ::= SEQUENCE {
    accessRightID AccessRightID ,--OPTIONAL,
    searchStrings SearchStrings ,
    creationTime CreationTime ,
    lastModifiedTime LastModifiedTime ,
    aPocHandling APocHandling,
    publicDomain String
}

AScls ::= SEQUENCE {
    accessRightID AccessRightID ,--OPTIONAL,
    creationTime CreationTime ,
    lastmodifiedTime LastModifiedTime
}

AScl ::= SEQUENCE {
    pocs ListOfM2mPoc,
    remTriggerAddr String ,--OPTIONAL,
    onlineStatus OnlineStatus ,
    serverCapability BOOLEAN ,
    link Link ,

```

```

        schedule                String                ,--OPTIONAL,
        expirationTime          ExpirationTime        ,
        accessRightID           AccessRightID         ,--OPTIONAL,
        searchStrings            SearchStrings        ,
        creationTime             CreationTime          ,
        lastmodifiedTime         LastModifiedTime      ,
        locTargetDevice          String                ,--OPTIONAL,
        mgmtProtocolType         MgmtProtocolList      ,--OPTIONAL,
        integrityValResults      String                ,--OPTIONAL,
        aPocHandling             String                ,--OPTIONAL,
        sclType                   SclType,
        announceTo               AnnounceTo
    }

AApplications ::= SEQUENCE {
    accessRightID           AccessRightID         ,--OPTIONAL,
    creationTime             CreationTime          ,
    lastModifiedTime         LastModifiedTime
}

AApplication ::= SEQUENCE {
creationTime CreationTime,
    expirationTime          ExpirationTime,
    accessRightID           AccessRightID         ,--OPTIONAL,
    searchStrings            SearchStrings,
    lastModifiedTime         LastModifiedTime,
--    announceTo             AnnounceTo,
    aPoC                    String                ,--OPTIONAL,
    aPoCPaths                String                ,--OPTIONAL,
    locRequestor             String                ,--OPTIONAL,
    referencePoint            ReferencePoint
}

AApplicationAnnnc ::=SEQUENCE{
    link                    Link,
    accessRightID           AccessRightID,      --    OPTIONAL,
    --announceTo             AnnounceTo          ,OPTIONAL,

```

```

        searchStrings      SearchStrings,
        expirationTime      ExpirationTime
    }

AAccessRights ::= SEQUENCE {
    accessRightID      AccessRightID      , -- OPTIONAL,
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime
}

AAccessRight ::= SEQUENCE {
    expirationTime      ExpirationTime      ,
    searchStrings      SearchStrings      ,
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime      ,
    announceTo      AnnounceTo      ,
    permissions      SeqOfPerm      ,
    selfPermissions      SeqOfPerm
}

AAccessRightAnnc ::= SEQUENCE {
    link      Link      ,
    accesRightID      AccessRightID      , -- OPTIONAL,
    announceTo      AnnounceTo      , -- OPTIONAL,
    searchStrings      SearchStrings      ,
    expirationTime      ExpirationTime
}

AContainers ::= SEQUENCE {
    accessRightID      AccessRightID      , -- OPTIONAL,
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime
}

AContainer ::= SEQUENCE {
    expirationTime      ExpirationTime      ,
    accessRightID      AccessRightID      , -- OPTIONAL,

```

```

        searchStrings      SearchStrings      ,
        creationTime       CreationTime       ,
        lastModifiedTime   LastModifiedTime   ,
        announceTo         AnnounceTo         ,
        maxNrOfInstances   INTEGER            ,
        maxByteSize        INTEGER            ,
        maxInstanceAge     INTEGER            ,
    }

AContainerAnnc ::=SEQUENCE{
    link          Link          ,
    accessRightID AccessRightID ,--OPTIONAL,
    announceTo     AnnounceTo    ,--OPTIONAL,
    searchStrings  SearchStrings ,
    expirationTime ExpirationTime
}

ALocationContainer ::=SEQUENCE{
    expirationTime      ExpirationTime      ,
    accessRightID       AccessRightID       ,--OPTIONAL,
    searchStrings       SearchStrings       ,
    creationTime        CreationTime        ,
    lastModifiedTime    LastModifiedTime    ,
    announceTo          AnnounceTo          ,
    maxNrOfInstance     INTEGER             ,
    maxByteSize         INTEGER             ,
    maxInstanceAge      INTEGER             ,
    locationContainerType String
}

ALocationContainerAnnc ::=SEQUENCE{
    link          Link          ,
    accessRightID AccessRightID ,--OPTIONAL,
    announceTo     AnnounceTo    ,--OPTIONAL,
    searchStrings  SearchStrings
}

```

```

AContentInstances ::=SEQUENCE{
    creationTime      CreationTime      ,
    lastModifiedTime  LastModifiedTime  ,
    currentNrOfInstances INTEGER          ,
    currentByteSize   INTEGER
}

AContentInstance ::=SEQUENCE{
    contentType      String      ,--OPTIONAL,
    contentSize      INTEGER          ,
    creationTime      CreationTime  ,
    lastModifiedTime  LastModifiedTime ,
    delayTolerance    INTEGER          ,--OPTIONAL,
    searchStrings     SearchStrings  ,--OPTIONAL,
    content           ContentData
}

AGroup ::=SEQUENCE {
    expirationTime      ExpirationTime      ,
    accessRightID        AccessRightID      ,--OPTIONAL,
    searchStrings        SearchStrings      ,
    creationTime         CreationTime       ,
    lastModifiedTime     LastModifiedTime   ,
    announceTo           AnnounceTo        ,
    memberType           String             ,
    currentNrOfMembers   INTEGER            ,
    members              SeqOfString,
    membersContentAccessRightID String ,--OPTIONAL,
    memberTypeValidated  BOOLEAN            ,
    consistencyStrategy String             OPTIONAL
}

AGroupAnnc ::=SEQUENCE{
    link              Link              ,
    accessRightID      AccessRightID    ,--OPTIONAL,
    announceTo         AnnounceTo       ,
    searchStrings      SearchStrings    ,

```

```

        expirationTime      ExpirationTime
    }

ASubscription ::= SEQUENCE{
    expirationTime      ExpirationTime,
    minimalTimeBetweenNotifications  INTEGER      ,--OPTIONAL,
    delayTolerance      TemporalVariable      ,--OPTIONAL,
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime      ,
    filterCriteria      FilterCriteria      ,--OPTIONAL,
    subscriptionType      ChannelType,
    contact      String      ,
    aggregateURI      String      ,--OPTIONAL,
    timeoutReason      String      ,--OPTIONAL,
    noRepresentation      BOOLEAN      ,--OPTIONAL,
    subscriberId String
--    subscriberId      CHOICE      {scIs      [0] ScIs,
--                                application [1]
Application}

}

AM2mPocs ::=SEQUENCE{
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime
}

AM2mPoc ::=SEQUENCE{
    contactInfo      String      ,
    expirationTime      ExpirationTime      ,
    onlineStatus      OnlineStatus,
    creationTime      CreationTime      ,
    lastModifiedTime      LastModifiedTime
}

AMgmtObjs ::=SEQUENCE{
    accessRightID      AccessRightID      ,
    creationTime      CreationTime      ,

```



```

        lastModifiedTime      LastModifiedTime
    }

    AMgmtObj ::= SEQUENCE {
        expirationTime      ExpirationTime      ,
        accessRightID       AccessRightID        ,
        searchStrings       SearchStrings        ,
        creationTime        CreationTime         ,
        lastModifiedTime    LastModifiedTime     ,
        moID                String               ,
        originalMO          String               ,
        moAttribute         String               , -- OPTIONAL,
        description         String               -- OPTIONAL
    }

    AParameters ::= SEQUENCE {
        originalMO          String               , -- OPTIONAL,
        accessRightID       AccessRightID        ,
        creationTime        CreationTime         ,
        lastModifiedTime    LastModifiedTime     ,
        moAttribute         String               -- OPTIONAL
    }

    AMgmtCmd ::= SEQUENCE {
        expirationTime      ExpirationTime      ,
        accessRightID       AccessRightID        ,
        searchStrings       SearchStrings        ,
        creationTime        CreationTime         ,
        lastModifiedTime    LastModifiedTime     ,
        description         String               , -- OPTIONAL,
        cmdType             String               ,
        execReqArgs         SeqOfString          , -- OPTIONAL,
        execEnable          String
    }

    AExecInstances ::= SEQUENCE {
        creationTime        CreationTime         ,

```

```
        lastmodifiedTime    LastModifiedTime
    }

AExecInstance                ::=SEQUENCE{
    expirationTime           ExpirationTime        ,
    accessRightID            AccessRightID          ,
    creationTime             CreationTime           ,
    lastModifiedTime         LastModifiedTime       ,
    execStatus               ExecStatus,
    execResult               String                ,
    execDisable              String                --OPTIONAL
}

AAttachedDevices             ::=SEQUENCE{
    accessRightID            AccessRightID          ,
    creationTime             CreationTime           ,
    lastmodifiedTime         LastModifiedTime
}

AAttachedDevice              ::=SEQUENCE{
    accessRightID            AccessRightID          ,
    creationTime             CreationTime           ,
    lastModifiedTime         LastModifiedTime
}

ANotificationChannels        ::=SEQUENCE{
    creationTime             CreationTime           ,
    lastModifiedTime         LastModifiedTime
}

ANotificationChannel         ::=SEQUENCE{
    channelType              ChannelType,
    contactURI               String                ,
    channelData              String                ,
    creationTime             CreationTime           ,
    lastModifiedTime         LastModifiedTime
}
```

```

ASubcontainers ::=SEQUENCE{
    accessRightID      AccessRightID      ,
    creationTime       CreationTime       ,
    lastModifiedTime   LastModifiedTime
}

ASclAnnCs           ::=SEQUENCE{
    accessRightID      AccessRightID      ,--OPTIONAL,
    creationTime       CreationTime       ,
    lastModifiedTime   LastModifiedTime
}

ASclAnnC            ::=SEQUENCE{
    link               Link               ,
    accessRightID      AccessRightID      ,
    searchStrings      SearchStrings      ,
    expirationTime     ExpirationTime
}

AAccessRightsAnnC ::=SEQUENCE {
link Link,
accessRightID AccessRightID,
announceTo AnnounceTo ,
searchStrings SearchStrings,
expirationTime ExpirationTime
}

AGroups ::= SEQUENCE {
accessRightID AccessRightID ,--OPTIONAL,
creationTime CreationTime,
lastModifiedTime LastModifiedTime
}

-- end of attribute declaration

-- Item sequences declaration

SclBase ::= SEQUENCE {

```

```

        attribute          ASclBase,
        scls                Scls,
        applications        Applications,
        containers          Containers,
        groups              Groups,
        accessRights        AccessRights,
        subscriptions        Subscriptions,
        discovery            Discovery
    }

Scls ::=SEQUENCE {
    attribute          AScls,
    scl                SeqOfScl,
    subscriptions      Subscriptions,
    mgmtObjs           MgmtObjs          --OPTIONAL
}

Scl ::=SEQUENCE {
    attribute          AScl,
    containers         Containers,
    groups             Groups,
    applications        SeqOfAppAnnc,
    accessRights        AccessRights,
    subscriptions       Subscriptions,
    mgmtObjs            MgmtObjs          ,--OPTIONAL,
    notificationChannels NotificationChannels,
    m2mPocs             M2mPocs          ,--
OPTIONAL,
    attachedDevices     AttachedDevices   ,--OPTIONAL,
    sclAnnCs            SclAnnCs          --OPTIONAL
}

Applications ::=SEQUENCE {
    attribute          AApplications,
    application         SeqOfApplication ,--OPTIONAL,
    applicationAnnc      SeqOfApplicationAnnc ,--OPTIONAL,
    subscriptions        Subscriptions,
    mgmtObjs             MgmtObjs          --OPTIONAL
}

```

```
}

Application ::=SEQUENCE {
    attribute                AApplication,
    containers                Containers,
    groups                   Group,
    accessRights             AccessRights,
    subscriptions            Subscriptions,
    notificationChannels     NotificationChannels
}

ApplicationAnnc ::= SEQUENCE {
    attribute                AApplicationAnnc,
    containers                Containers,
    groups                   Groups,
    accessRights             AccessRights
}

AccessRights ::= SEQUENCE {
    attribute                AAccessRights,
    accessRight              SeqOfAccessRight,
    accessRightAnnc          SeqOfAccessRightAnnc,
    subscriptions            Subscriptions
}

AccessRight ::=SEQUENCE {
    attribute                AAccessRight,
    subscriptions            Subscriptions
}

AccessRightAnnc ::= SEQUENCE {
    attribute                AAccessRightAnnc
}

Containers ::= SEQUENCE {
    attribute                AContainers,
    container                SeqOfContainer,
```

```

        containerAnnc          SeqOfContainerAnnc,
        locationContainer      SeqOfLocContainer  ,--OPTIONAL,
        locationContainerAnnc  SeqOfLocContainerAnnc ,--OPTIONAL,
        subscriptions          Subscriptions
    }

Container      ::= SEQUENCE {
    attribute          AContainer,
    --deny loop
--    subcontainers      Subcontainers,
    contentInstances   ContentInstances,
    subscriptions      Subscriptions
}

ContainerAnnc ::= SEQUENCE {
    attribute          AContainerAnnc
}

LocationContainer ::= SEQUENCE {
    attribute          ALocationContainer,
    contentInstances   ContentInstances,
    subscriptions      Subscriptions
}

LocationContainerAnnc ::=SEQUENCE {
    attribute          ALocationContainerAnnc
}

ContentInstances ::=SEQUENCE {
    attribute          AContentInstances,
    contentInstance    SeqOfContentInstance ,--OPTIONAL,
    latest             ContentInstance      ,--OPTIONAL,
    oldest             ContentInstance      ,--OPTIONAL,
    subscriptions      Subscriptions
}

ContentInstance ::=SEQUENCE {

```

```

    attribute                AContentInstance,
    content                   Content
}

Groups ::=SEQUENCE{
    attribute                AGroups,
    group                    SeqOfGroup,
    groupAnnc                SeqOfGroupAnnc,
    subscriptions            Subscriptions
}

Group ::=SEQUENCE {
    attribute                AGroup,
    members                  Members,
    membersContent           MembersContent,
    subscriptions            Subscriptions
}

GroupAnnc ::=SEQUENCE {
    attribute                AGroupAnnc
}

MembersContent ::= SEQUENCE {
    ressource                SeqOfString
}

Subscriptions ::=SEQUENCE{
    subscription             SeqOfSubscription
}

Subscription ::=SEQUENCE {
    attribute                ASubscription
}

M2mPocs ::=SEQUENCE{
    attribute                AM2mPocs,
    m2mPoc                  SeqOfM2mPoc --OPTIONAL
}
```

```
}

M2mPoc                ::= SEQUENCE{
    attribute            AM2mPoc
}

MgmtObjs              ::= SEQUENCE{
    attribute            AMgmtObjs,
    mgmtObj              SeqOfMgmtObj,
    mgmtCmd              SeqOfMgmtCmd,
    subscriptions        Subscriptions
}

MgmtObj               ::= SEQUENCE{
    attribute            AMgmtObj,
    parameters           SeqOfParameters,
    subscriptions        Subscriptions
}

Parameters            ::=SEQUENCE{
    attribute            AParameters,
    --deny loop
--    parameters         SEQUENCE OF Parameters,
    subscriptions        Subscriptions
}

MgmtCmd               ::=SEQUENCE{
    attribute            AMgmtCmd,
    execInstances        SeqOfLink,
    subscriptions        Subscriptions
}

ExecInstances         ::= SEQUENCE{
    attribute            AExecInstances,
    execInstance         SeqOfExecInstance,
    subscriptions        Subscriptions
}
```



```
ExecInstance      ::= SEQUENCE{
    attribute          AExecInstance,
    subscriptions      Subscriptions
}

AttachedDevices   ::= SEQUENCE{
    attribute          AAttachedDevices,
    attachedDevice     SeqOfAttachedDevice ,--OPTIONAL,
    subscriptions      Subscriptions
}

AttachedDevice    ::= SEQUENCE{
    attribute          AAttachedDevice,
    mgmtObjs           MgmtObjs,
    subscriptions      Subscriptions
}

NotificationChannels ::=SEQUENCE{
    attribute          ANotificationChannels,
    notificationChannel SeqOfNotificationChannel
}

NotificationChannel ::=SEQUENCE{
    attribute          ANotificationChannel
}

Discovery          ::=SEQUENCE{
    result             SeqOfString
}

Subcontainers      ::=SEQUENCE{
    attribute          ASubcontainers,
    container          Container,
    subscriptions      Subscriptions
}
```

```
SclAnncs ::=SEQUENCE{
    attribute          ASclAnncs,
    sclAnnnc           SeqOfSclAnnnc,
    subscriptions      Subscriptions
}

SclAnnnc ::=SEQUENCE {
    attribute          ASclAnnnc,
    containers         Containers,
    groups             Groups,
    applications        Applications,
    accessRights        AccessRights
}

AccessRightsAnnnc ::= SEQUENCE {
attribute AAccessRightAnnnc
}

-- end of Item sequence declaration

END
```

3.3 - Abbreviations

In order to fully understand this rapport, some abbreviations are provided here.

(for more information, please refer to the ETSI TR102725v010101p document).

- CoAP Constrained Application Protocol
- CRUD Create, Retrieve, Update and Delete
- DA Device Application
- D'A Device' Application
- D/GA Device or Gateway Application
- DA/GA Device or Gateway Application
- D/GSCL Device/Gateway Service Capabilities Layer
- DSCL Device Service Capabilities Layer
- GA Gateway Application

- GSCL Gateway Service Capabilities Layer
- M2M Machine to Machine
- M2MPoC M2M Point of Contact
- mIa M2M application Interface
- mId M2M device Interface
- MSBF M2M Service Bootstrap Function
- NA Network Application
- NSCL Network Service Capabilities Layer
- PoC Point of Contact
- SCL Service Capability Layer
- URI Uniform Resource Identifier
- URL Uniform Resource Locator

4 -CONCLUSION

The ETSIM2M is a reliable, secured and scalable protocol. It can handle multiple purposes on multiple scale levels. When the ETSI started developing this protocol, they wanted it to be as flexible as possible.

Machine to Machine communication is a fast growing sector where the technical support already exists (automated message routing, wireless communications, high data volume transfer) however now there is a high need of standards in order to provide a common solution to every industrial part. The ETSI M2M provide all those features by creating a deep and complete protocol handling most of the desired features that a M2M communication protocol needs.

Using RTDS to create an example allows having a simpler graphical interface for all the different elements that interact. The top level view gives a glimpse at what are the different parts of the system, each block give more detail about the composition of it. For even more information, the processes tell the execution pattern and the manipulated variables. It is also possible to see an graphical trace of execution of the system in order to verify step by step what process communicates, what variable are exchanged and if they received or ignored incoming messages.

Alphabetical Index

accessRight.....	18, 26
accessRightID.....	19, 27
appAskDeviceValue.....	29, 34
application.....	23
ASN1.....	8
asyn_rdy.....	15
backOff.....	33
channelType.....	14 sv, 20 sv, 29
charstring.....	8
confCreateDA.....	17, 38
confCreateNA.....	36, 38
createApp.....	36
createAppRegistration.....	23, 28
createDA.....	12, 20
createdDevices1st.....	37
createDevice1st.....	5
createdGSCL.....	19, 27
createdNA.....	28, 34
createGSCL.....	19, 27
createNA.....	27 sv, 33
createTimeout.....	33
DA.....	4 sv
deleteDA.....	16, 24
deleteNA.....	31, 35
deleteNASub.....	23 sv, 31
deleteScl.....	24, 32
DeletingApp.....	31
device.....	29
deviceChangeValue.....	15 sv
deviceName.....	23
deviceReset.....	12
deviceType.....	8
envCreateDA.....	17
envDeleteDA.....	16
envDeleteNA.....	35
ETSI.....	57
ETSI M2M.....	4, 6, 12
ETSIM2M.....	57
failure.....	19, 22 sv, 27 sv, 33, 35
gateway.....	4, 23

getDAList.....	17
getDeviceFromApp.....	36
getNAList.....	36
GSCL.....	5, 18
http://pragmadev.m2m.com.....	4
IA5String.....	8
intArray.....	8
lot.....	4
lib.....	8
longPolling_rdy.....	15
M2M.....	4
m2mPragmaBox.....	4
MAX_APP.....	8
MAX_DEVICE.....	7 sv, 16, 38
MAX_FAIL.....	8
MAX_TRY.....	8, 12, 32
NA.....	4 sv
nameArray.....	8
NB_INITIAL_DEVICE.....	8
Network applications.....	4
normal_rdy.....	15
notifyApp.....	21, 30
NSCL.....	4 sv, 18
pDA.....	10
pGSCL.....	17
pidArray.....	8
pollWait.....	34
providedDAList.....	17
providedDevices.....	35
providedNAList.....	36
providedValue.....	30, 34 sv
ready.....	19
Restful.....	4
retreiveDeviceValue.....	21
retrieveAccessRightID.....	19, 27
retrieveDeviceValue.....	22, 29
retrieveSearchString.....	19, 27
retrieveValue.....	15
RTDS.....	4
SCL.....	18
SDL.....	8
searchString.....	18, 26

searchStrings.....	19, 27
sendDeviceList.....	35 sv
SendDevicesList.....	35
setOffline.....	16
setOffline.....	16
smallRandom.....	8
subscription.....	13 sv, 23
subscriptionType.....	22
success.....	13 sv, 20, 24, 28, 31, 35
tNotify.....	5, 14 sv
tRegisterTimeout.....	12
unsubscription.....	16, 24
updateScl.....	20, 30
updateValue.....	8, 13 sv, 20 sv
waitingSubs.....	16



This document was generated with RTDS © (Real Time Developer Studio ©) a tool from

