ARM® SENSINODE
IoT Technology

NanoService

Version 2.0

NanoService Platform User Guide

Confidential

ARM®

ARM DUI 0827A (ID081214)

NanoService
NanoService Platform User Guide

Release Information

Change History

Date	Issue	Confidentiality	Change
August 2014	A	Confidential	NanoService Platform User Guide for NanoService v2.0 Release

Proprietary Notice

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
NanoService NanoService Platform User Guide

Preface

This preface introduces the *NanoService NanoService Platform User Guide* and comprises the following sections:

About this book

This book provides support for third-party developers who are using the NanoService NanoService Platform User Guide.

Intended audience

This book has been written for experienced software developers to assist with using the Sensinode software product range.

Using this book

This book is organized into the following chapters:

Chapter 1 *NanoService Security*

Read this to learn more about the NanoService Security solution.

Chapter 2 *Installation*

Read this for NanoService platform installation instructions.

Chapter 3 *M2M Interfaces*

Read this to learn more about the M2M interfaces.

Chapter 4 *Web Interfaces*

Read this to learn more about the web interfaces.

Chapter 5 *NanoService C Device Library*

Read this to learn more about the C device library.

Chapter 6 *NanoService Java Device Library*

Read this to learn more about the Java device library.

Chapter 7 *NanoService Java SDK*

Read this to learn more about the NSP Java SDK.

Chapter 8 *NanoService Java Use Cases*

Read this to see example use cases.

Chapter 9 *Quick Start Guide for the Lighting Example*

Read this to install and use the supplied Lighting example.

Chapter 10 *Quick Start Guide for the Connected Home Example*

Read this to install and use the supplied Connected Home example.

Typographical conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace` (underlined: `mono`)	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists related publications by ARM and by third parties.

See Infocenter http://infocenter.arm.com for access to ARM documentation.

ARM Sensinode related publications

[6LoWPAN-ND]	Network Discovery Optimization for IPv6 over Low power Wireless Personal Area Networks
[CoAP]	Constrained Applications Protocol http://tools.ietf.org/html/draft-ietf-core-coap-18
[FCC]	Electronic Code of Federal Regulations, Title 15, "Radio Frequency Devices" subpart 247, "Operation within the bands 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz" http://www.ecfr.gov/cgi-bin/text-idx?c=ecfr&tpl=/ecfrbrowse/Title47/47cfrv1_02.tpl
[IEEE]	IEEE Standards Association http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf (Membership may be required to access this information)
[NIST]	Smart Grid at the National Institute of Standards and Technology http://www.nist.gov/smartgrid/
[SKYWORK]	SE2431L, "2.4 GHz ZigBee/802.15.4 Front End Module" http://www.skyworksinc.com/Product.aspx?ProductID=933

Glossary

The *ARM Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM Glossary* http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html

ARM Sensinode-specific glossary

6LBR	6LoWPAN Border Router
6LN	6LoWPAN Node
6LoWPAN	IPv6 over Low power Wireless Personal Area Network
6LR	6LoWPAN Router
ACK	Acknowledgement
AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
BSD	Berkeley Software Distribution
CBC	Cipher Block Chaining (Message Authentication Code)
CCM	Counter Mode with CBC-MAC; see CBC
CoAP	Constrained Application Protocol
DAD	Duplicate Address Detection
DODAG	Destination Oriented Directed Acyclic Graph
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
ECC	Elliptic Curve Cryptography
eDTLS	Enhanced DTLS
FCC	Federal Communications Commission
FHSS	Frequency-hopping Spread Spectrum
GCC	GNU Compiler Collection
GTS	Guaranteed Time Slot
HAL	Hardware Abstraction Layer
IAR	Ingenjörsfirman Anders Rundgren (Embedded Workbench)
ICMP	Internet Control Message Protocol
ID	Identifier
IDE	Integrated Development Environment
IDP	Integrated Development Platform
IETF	Internet Engineering Task Force
I/O	Input/Output
IP	Internet Protocol
IRQ	Interrupt Request

LQI	Link Quality Indicator
M2M	Machine to Machine
MAC	Media Access Control (layer)
MAC	Message Authentication Code
MLE	Mesh Link Establishment
ND	Neighbor Discovery
NIST	National Institute of Standards and Technology
NWK	Network
OSI	Open Systems Interconnection
PA	Power Amplifier
PAN	Personal Area Network
PANA	Protocol for carrying Authentication for Network Access
PHY	Physical (layer)
PIN	Personal Identification Number
PIO	Prefix Information Option
PKI	Public Key Infrastructure
PSK	Pre-Shared Key
RA	Router Advertisement
RAM	Random Access Memory
REST	REpresentational State Transfer
RF	Radio Frequency
RPL	Routing Protocol for Low power and Lossy networks
RS	Router Solicitation
RSSI	Received Signal Strength Indication
RTC	Real-time Clock
RTOS	Real-time Operating System
RTT	Round-Trip-Time
RX	Receive
SAM	Smart ARM Microcontrollers
SPI	Serial Peripheral Interface
TC	Timer/Counter
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access

TLS	Transport Layer Security
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms if appropriate.

Feedback on content

If you have any comments on this book, send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0827A
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
NanoService Security

This chapter discusses the architecture, features, and use of the Sensinode NanoService™ Security solution. It comprises the following sections:

1.1 Introduction

Recognizing that web applications based on the *REpresentational State Transfer* (REST) architecture have become ubiquitous, backend *Machine-to-Machine* (M2M) platforms must be based on the REST architecture of the Web to reduce the time taken to develop and deploy applications for accessing M2M networks. The following is an overview of the NanoService Security solution for providing end-to-end authentication, integrity and confidentiality between M2M devices, the NanoService Platform and web applications.

The NanoService solution leverages the power of the web architecture for developing and deploying M2M systems efficiently and securely. The solution consists of software for devices, backend servers and web applications that together form an end-to-end platform as shown in Figure 1-1.

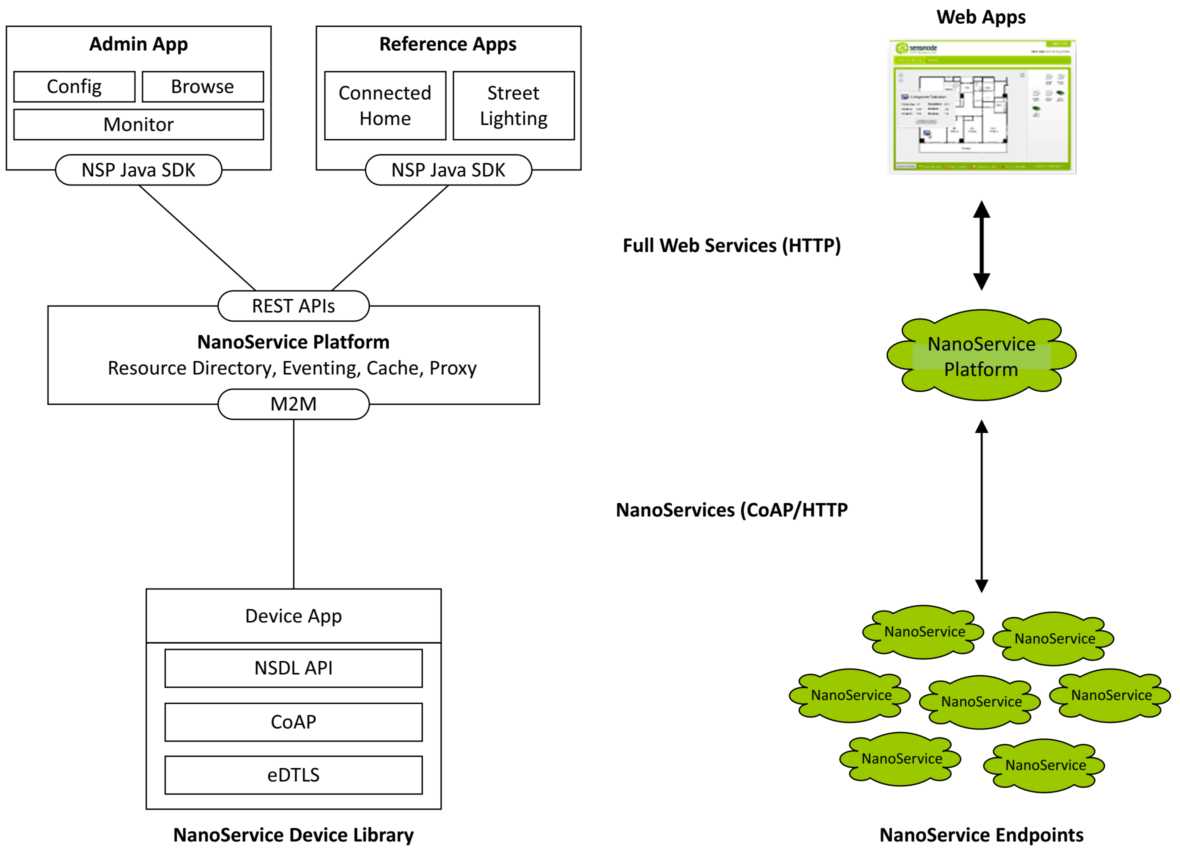


Figure 1-1 NanoService overall architecture and components

1.2 NanoService Security

NanoService Security leverages the power and flexibility of public key cryptography with innovative *Transport Layer Security* (TLS) technology to take full advantage of the NanoService web architecture and efficiency. The architecture of the NanoService Security solution is shown in Figure 1-2.

Security between M2M devices and NanoService Platform is provided on the M2M interface using Sensinode's *enhanced Datagram TLS* protocol (eDTLS) to provide mutual authentication, integrity protection and confidentiality for all CoAP traffic between a device and the platform. Support for other security protocols including full DTLS and TLS are on the NanoService roadmap.

NanoService Platform provides support for simple whitelist configuration for authentication of M2M devices. M2M interface security can also be fully load balanced between clusters of NSP nodes for system scalability.

Security between web applications and NanoService Platform is performed on the web interface using TLS for all HTTP traffic (HTTPS). Authentication is performed based on whitelist configuration, and domain-based access control is provided on devices and resources available through the platform. This authentication model removes the need for the web application to manage device provisioning and credential authentication.

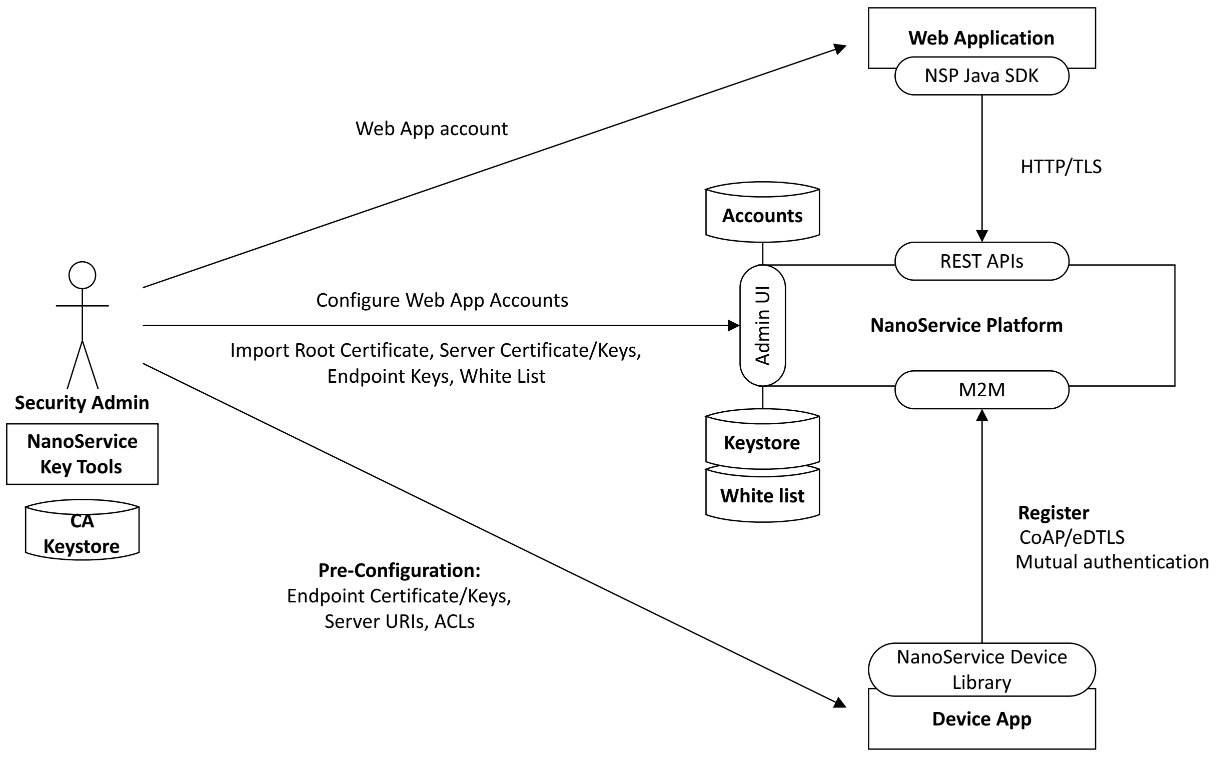


Figure 1-2 NanoService security architecture

1.3 M2M interface security

Security between M2M devices and NanoService Platform is achieved using Sensinode's enhanced DTLS (eDTLS) technology which provides TLS protocol primitives and cipher suite technology with less overhead and state for M2M use. Security on the M2M device is provided by the NanoService Device Library which includes eDTLS protocol and crypto libraries for embedded microcontrollers. NanoService Platform includes acts as the security endpoint and provides white list and load balancing integration.

NanoService Security on the M2M interface provides three different security modes for keying and authentication:

Pre-Shared Key

The device and platform are provisioned with a unique key and key identity shared by both entities. This key is used to generate a session key during the security handshake. Mutual authentication is performed using the key identity.

Raw Public Key

The device and platform are each provisioned a public/private key pair. These key pairs are used to perform an ECC asymmetric key exchange generating a session key. Mutual authentication is performed using the public key identity. This mode is not yet supported in this release of NanoServices.

Certificate The device and platform are each provisioned with an X.509 certificate. The public/private key pairs associated with the certificate are used to perform an ECC asymmetric key exchange generating a session key. Mutual authentication is performed using an identity included in the certificate.

1.4 Web interface security

NanoService Platform provides HTTP REST interfaces for use by web applications. Security for these applications is provided using TLS based HTTPS with Certificates. User based authentication is performed on a domain model. The access control of a user to one or more domains is configurable.

1.5 Keys and certificates

NanoService Security uses a key material pre-provisioning model, where any needed keying material is pre-installed in the server (NanoService Platform) and in each device (for use with NanoService Device Library). This keying material is used only for the purpose of the TLS handshake, and is never used directly for encryption or integrity. Temporary session keys are always generated as a result of the TLS handshake, and are used for the duration of the session.

This section describes the cipher suites supported by each security mode of NanoServices, how keys are generated and used, and how X.509 Certificates are used.

1.5.1 Pre-Shared Key mode

In *Pre-Shared Key* (PSK) mode, a 128-bit key is pre-provisioned on NanoService Platform and each endpoint for use in the TLS handshake. As this is a symmetric key, the key of each endpoint must also be configured in the authentication file (whitelist) of NanoService Platform. This key is used to authenticate the device, and if it can access that server.

Cipher suites

In this mode, the cipher suite `TLS_PSK_WITH_AES_128_CCM_8` is currently supported. This cipher suite is required by the CoAP specification. This cipher is well suited to simple microcontrollers, and takes advantage of the *Advanced Encryption Standard* (AES) hardware engines available on many chips. NanoService Device Library takes advantage of hardware AES on supported platforms.

Generating keys

The generation of a PSK does not require any special tools, it is simply a random 128-bit number. However, each PSK used with a NanoService deployment must be unique. OpenSSL can be used to randomly generate keys.

1.5.2 Certificate mode

In Certificate mode, a unique X.509 ECC Certificate and private key are generated for each server and device. This so-called asymmetric key pair is used in the TLS handshake to generate temporary session keys for use during that TLS session. Private keys are never shared outside of a server or device.

Unlike PSK mode, there is no need to share certificates between NanoService Platform and endpoints. Instead, authentication of devices is bound to the Endpoint Name embedded in each certificate. NanoService Platform needs to be configured with the list of Endpoint Names that are allowed to register to that server, and verifies that the registering endpoint really is who it claims to be. Furthermore, the root certificate may be used by both ends of the TLS handshake to verify the signature of a certificate.

Cipher suites

This security mode currently supports the cipher `TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8` with 256-bit ECC keys and SHA-256 signature. This cipher takes advantage of the same AES hardware available on many devices.

X.509 Certificate specification

NanoService Security supports X.509v3 Certificates with suitable public key and signature algorithms for the cipher suite used. NanoService does not currently support certificate chains, however all server and endpoint certificates are directly signed by the root certificate.

The following fields are used in a NanoService root certificate by default:

- `Issuer Country`.
- `Issuer Organization`.
- `Issues Common Name`.

The following is an example of a root certificate generated by the NanoService Certificate Tools:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1075672988 (0x401d779c)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=FI, O=Sensinode Ltd, CN=Test CA
        Validity
            Not Before: Mar 27 11:30:52 2013 GMT
            Not After : Mar 26 11:30:52 2017 GMT
        Subject: C=FI, O=Sensinode Ltd, CN=Test CA
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
            EC Public Key:
                pub:
                    04:a1:cc:c1:02:0e:74:7b:25:47:ee:03:aa:80:f6:
                    0f:e7:6f:25:94:91:02:1a:46:6f:9f:bd:d5:85:11:
                    7b:52:8a:a4:0f:71:47:eb:ea:f1:fd:be:a8:9b:92:
                    7f:34:5a:73:d9:77:f7:5b:90:56:66:02:75:49:f2:
                    ba:b6:4e:a4:f2
                ASN1 OID: prime256v1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                37:77:2F:78:CA:95:80:A0:B1:CD:91:70:D0:84:D0:F0:BC:2A:3A:35
    Signature Algorithm: ecdsa-with-SHA256
        30:44:02:20:06:e4:64:29:34:88:fe:d3:ef:a9:54:71:b3:44:
        14:a8:fd:30:27:ad:c8:2a:4c:e9:07:02:46:34:7e:c6:9f:dd:
        02:20:48:be:20:66:f5:d2:12:1b:a3:97:57:95:bd:85:c7:37:
        b3:bd:c2:0c:c5:ab:ef:94:ed:b3:c3:49:8e:29:d1:59
```

The following fields are used in a NanoService server certificate by default:

`Subject Country`

Typically the same as the `Issuer Country`.

`Subject Organization`

Typically the same as the `Issuer Organization`.

The following is an example of a server certificate generated by the NanoService Certificate Tools:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1793451298 (0x6ae5e522)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=FI, O=Sensinode Ltd, CN=Test CA
        Validity
            Not Before: Mar 27 11:30:54 2013 GMT
            Not After : Mar 27 11:30:54 2015 GMT
        Subject: C=FI, O=Sensinode Ltd
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
            EC Public Key:
                pub:
                    04:1d:a3:25:8e:f8:77:ab:44:a4:98:97:d3:19:77:
                    66:46:ac:9e:d6:5a:ff:f4:37:c0:27:06:01:15:c6:
                    21:33:8d:e3:ed:5d:40:41:ab:66:49:8e:59:0a:2c:
                    79:f4:08:1d:2a:a7:fa:42:ab:5b:65:00:8a:63:af:
                    8d:35:8f:bd:7f
                ASN1 OID: prime256v1
    Signature Algorithm: ecdsa-with-SHA256
        30:44:02:20:45:88:f5:db:39:5f:aa:fc:62:9d:b6:00:f6:26:
        ae:ca:5f:ec:9c:1f:b6:45:8b:1e:5c:e2:51:f7:a8:a4:b4:cc:
        02:20:65:89:7b:1b:58:bf:34:6d:8f:6e:da:8f:03:cd:1b:5d:
        29:ad:51:d3:e0:54:b6:7d:e9:13:2a:80:32:3d:ca:14
```

The following fields are used in a NanoService endpoint certificate by default:

`Subject Country`

Typically the same as the `Issuer Country`.

`Subject Organization`

Typically the same as the `Issuer Organization`.

`Subject Common Name`

The endpoint name of the device in the form `endpoint.domain`. This endpoint name must be used by this node when registering with the NSP.

The following is an example of an endpoint certificate generated by the NanoService Certificate Tools. In the example below the endpoint name is `node-001`:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 646085195 (0x26827a4b)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=FI, O=Sensinode Ltd, CN=Test CA
        Validity
            Not Before: Mar 27 11:42:58 2013 GMT
            Not After : Mar 27 11:42:58 2015 GMT
        Subject: CN=node-001, C=FI, O=Sensinode Ltd
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
            EC Public Key:
                pub:
                    04:a2:2a:17:76:d9:f3:03:bc:0b:91:62:c9:ed:2d:
                    16:f3:c7:75:77:e5:bf:22:56:da:8c:6f:e8:ee:64:
                    fe:3e:ea:89:72:3d:9d:a8:48:b9:64:6a:83:9b:69:
                    36:dc:32:41:28:0d:dd:7d:87:b7:39:6a:75:ee:1c:
                    1c:e5:27:49:e5
                ASN1 OID: prime256v1
    Signature Algorithm: ecdsa-with-SHA256
        30:46:02:21:00:c0:96:25:67:22:28:94:e2:ea:b1:b9:99:16:
        c3:27:c3:12:4c:33:3c:80:cb:04:f4:1c:1f:57:ee:c5:fe:48:
```

```
e7:02:21:00:f6:6e:35:c0:c8:ab:4d:84:7a:b6:44:f0:73:7c:
d1:10:68:d5:67:73:c1:20:a9:b7:68:4d:17:cf:7b:b3:57:24
```

Generating certificates

NanoService Security includes a tool set for generating, managing and exporting private keys and certificates for servers and endpoints. For detailed information, see the accompanying documentation of the NanoService Certificate Tools component included in NanoService Enterprise.

1.6 Using NanoService Security

This section gives an overview of how to set up and use NanoService Security in practice using PSK mode.

1.6.1 Node Emulator

Node Emulator has built-in support for PSK and certificate modes, and when started with the `-key` or `-keyfile` command-line parameters, will automatically perform a handshake with the NanoService Platform.

The `-key` parameters pass the key ID and key to be used in the handshake. This key ID and key pair MUST be configured in the `credentials.csv` file of the NanoService Platform for the handshake to work.

An example of starting the Node Emulator in PSK mode:

```
java -jar ../lib/node-emulator-*.jar -def ../conf/power-node.json,1 -nsp 127.0.0.1
-key 0F0F:31323334353637383930313233343536
```

1.6.2 NanoService Platform

NanoService Platform supports PSK and certificate client modes of eDTLS. The properties file `nsp.properties` contains parameters for eDTLS. The file `credentials.csv` contains an example of PSK definition that will be taken into use after enabling the transport.

1.6.3 NSDL C

NSDL C includes support for PSK and certificate client modes of eDTLS. This functionality is provided by the libEDTLS library.

The library is included as a standard C library in your project:

1. To initialize the library, use the function call `sn_edtls_libraray_initialize()`.
2. To configure the PSK, use the call `edtls_pre_shared_key_set()`.
3. Finally, to connect to another eDTLS endpoint, use the `sn_edtls_connect()` function call.
4. When a successful connection has been made, use the `sn_edtls_write_data()` and `sn_edtls_parse_data()` calls to send and receive data.

For more information, see the *NSDL C* and *Doxygen* documentation.

Chapter 2
Installation

This chapter describes the installation, configuration, and interfaces of the NanoService Platform.

It comprises the following sections:

2.1 Installation

This section describes the installation requirements and options.

2.1.1 Requirements

NanoService Platform (NSP) has the following requirements:

- A single server or virtual machine for the installation:
  - The NSP has been tested and verified on CentOS 5.6.
  - The system requires a minimum of 1GB of memory per component run.
- Sun Java JVM v1.7.
- Java Cryptography Extension, if using eDTLS. See:
  `http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html`

2.1.2 Networking

Table 2-1 lists the default ports used by the NSP platform that must be opened from the system firewall for the platform to function properly. Ports are system component dependent, and if the system is distributed to be run on several machines, the ports must be opened accordingly. You can also change the ports in the configuration file.

Table 2-1 NSP default ports

Protocol	Port	Direction	Explanation
UDP	5683	IN/OUT	Non-encrypted CoAP interface
UDP	5684	IN/OUT	Encrypted CoAP interface (disabled by default)
TCP	5683	IN/OUT	Non-encrypted CoAP over TCP interface (disabled by default)
TCP	5684	IN/OUT	Encrypted CoAP over TCP interface (disabled by default)
TCP	8080	IN	NSP HTTP server port
TCP	6666	IN	NSP JMX monitoring port
TCP	9510	OUT	Terracotta server

2.1.3 Installing NSP

Copy the NSP installation package to your home directory and uncompress it:

```
tar -xvzf NSP-<version information>.tar.gz
```

Table 2-2 lists the package contents.

Table 2-2 Installation package contents

Path	Description
`/`	Contains `README.txt` and `CHANGELOG.txt` files. `README.txt` includes information on the package contents and instructions on how to run it. `CHANGELOG.txt` has information on the release versioning and changes in relation to the previous version.
`/bin`	Contains the scripts to run and configure the installation package content.
`/conf`	Contains configuration data for the installation package.
`/conf/templates`	Contains endpoint configuration template files. Files with `.json` extension are parsed as template information for the server.
`/lib`	Contains binary jar files of the installation package.
`/lib/ext`	Contains external dependencies of the installation package.
`/log`	Contains daily rolling log files.
`/license`	Contains the license files associated with the software release.

2.2 Configuration

This section describes how to run and configure the NSP.

2.2.1 Running NSP as a single node

To run the NSP as a single node, go to the NSP installation directory:

```
cd nsp-enterprise-<version information>/bin
./runNSP.sh
```

2.2.2 Running NSP as a cluster node

Running the NSP as a cluster node requires a store that handles memory distribution across multiple servers. To manage the in-memory data, you can use Terracotta BigMemory MAX.

To run the NSP as a cluster node, download and run the Terracotta BigMemory MAX:

1. Download a free version of the Terracotta BigMemory MAX from:

   `http://terracotta.org/products/bigmemory`

2. Copy the following libraries into the NSP's `nsp-enterprise-<version information>/lib/terracotta/` folder:

   ```
   bigmemory-max-4.x.x/apis/ehcache/lib/ehcache-ee-2.7.x.jar
   bigmemory-max-4.x.x/apis/toolkit/lib/terracotta-toolkit-runtime-ee-4.x.x.jar
   bigmemory-max-4.x.x/common/lib/bigmemory-4.x.x
   ```

3. Copy the license key `terracotta-license.key` into the NSP's `nsp-enterprise-<version information>/conf/` folder.

4. Prepare the Terracotta configuration file `tc-config.xml`, for example:

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
   <tc:tc-config xmlns:tc="http://www.terracotta.org/config"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.terracotta.org/
   schema/terracotta-6.xsd">

      <servers>
         <server>
             <!-- Enable BigMemory on the server. -->
             <offheap>
               <enabled>true</enabled>
               <maxDataSize>5g</maxDataSize>
             </offheap>
         </server>
     </servers>
     <clients>
        <logs>../log</logs>
     </clients>
   </tc:tc-config>
   ```

5. Run the terracotta server:

   ```
   cd bigmemory-max-4.x.x/server/bin
   ./start-tc-server.sh tc-config.xml
   ```

   For more information on the Terracotta BigMemory MAX, see `http://terracotta.org/documentation/4.0/bigmemorymax/overview`.

6. To start the NSP, go to the NSP installation directory and run the NSP, providing the terracotta address:

```
cd nsp-enterprise-<version information>/bin
./runNSP-tc.sh 127.0.0.1:9510
```

2.2.3 Multiple NSP node configuration in a cluster mode

Setting up a cluster with multiple NSP nodes requires proper network configuration and a load balancer to handle traffic.

The load balancer listens on the port where the clients connect to access services. The load balancer forwards client requests to one of the backend servers that replies to the load balancer. This way the load balancer can reply to the clients without them knowing of the internal separation of functions. This also prevents the clients from contacting the backend servers directly, which brings security benefits.

Figure 2-1 is an example of a setup and message flows in a clustered environment:

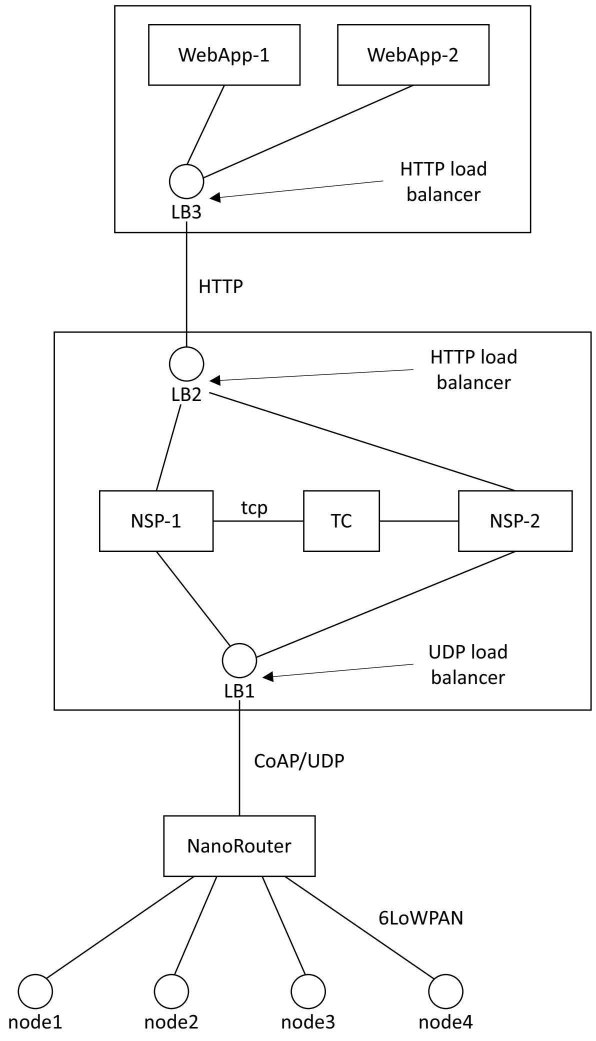


Figure 2-1 Multiple NSP node setup in cluster mode

Where:

WebApp-1, WebApp-2

Clustered nodes for web applications. A shared database or a cache is required between the nodes.

LB3	HTTP load balancer for incoming requests from the NSP (notifications).
LB2	HTTP load balancer for incoming requests from the web applications. No sticky session required.
LB1	UDP load balancer for incoming data from the nodes.
NSP-1, NSP-2	Clustered NSP nodes.
TC	Terracotta server. Each NSP node must be connected to the same Terracotta server.

Figure 2-2 introduces scenario 1, where a web application sends a request:

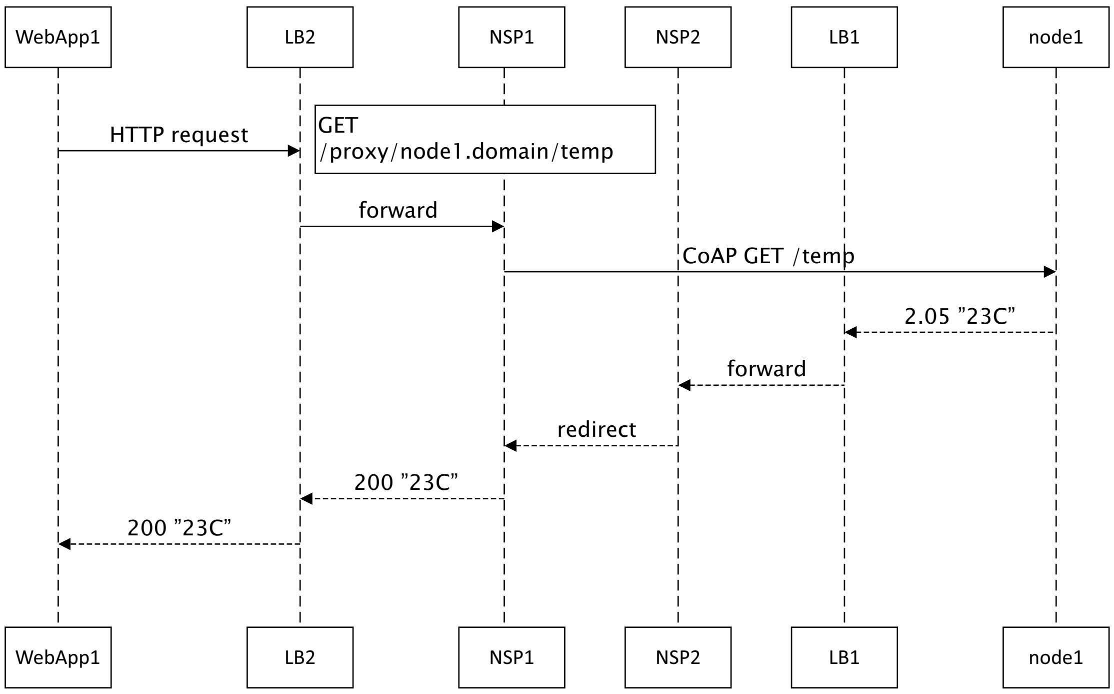


Figure 2-2 Scenario 1: Request from a web application

The message flow is as follows:

1. A web application sends a proxy request to node1.
2. The load balancer forwards the request to the NSP.
3. The NSP sends a CoAP request to the node.
4. The node responses with a CoAP message containing a resource value to the load balancer.
5. The load balancer forwards the message to one NSP instance.
6. If the NSP instance is not the originating message handler, the message is redirected internally to the correct NSP node via a shared cache.
7. The NSP responses to the web application.

Figure 2-3 on page 2-7 introduces scenario 2, where a notification comes from a node:

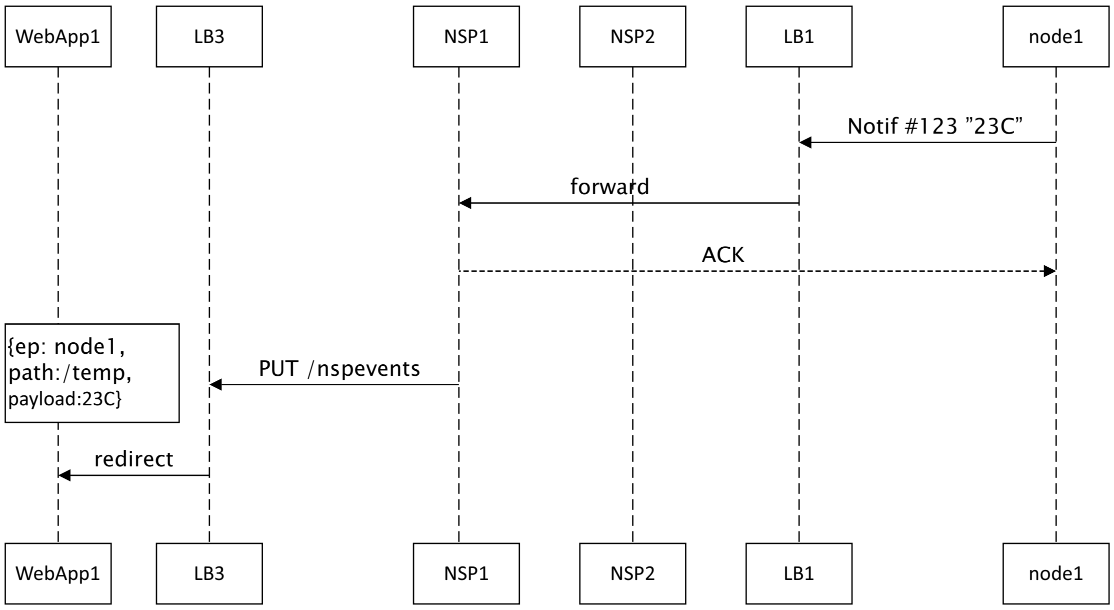


Figure 2-3 Scenario 2: Notification from a node

The message flow is as follows:

1. Node1 sends a notification that a web client has subscribed to the load balancer.
2. The load balancer forwards the message to an NSP instance.
3. The NSP responses with a CoAP ACK.
4. The NSP sends the notification to the web application.

Network and load balancing configuration

Figure 2-4 on page 2-8 is a deployment example that was created using Barracuda 440 load balancer with two NSP nodes. The Barracuda was set up as a "Two-armed route-path" configuration. Both CoAP UDP IPv6 and TCP IPv4 traffic was balanced with the same Barracuda load balancer.

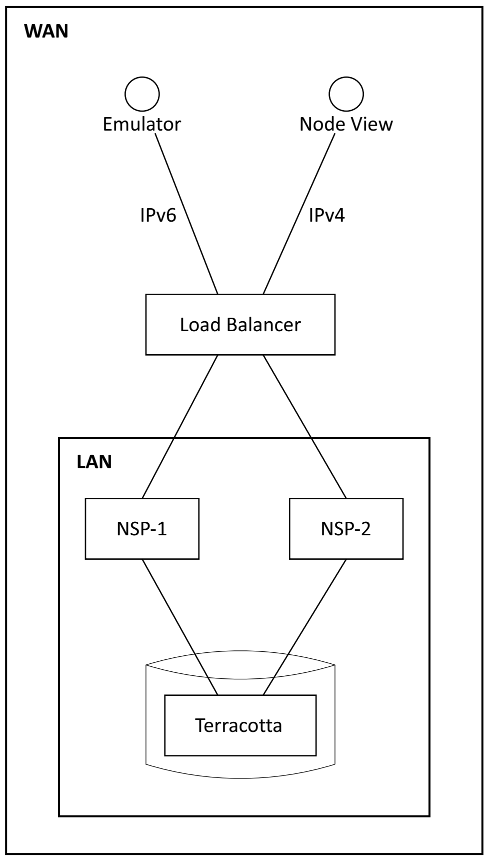


Figure 2-4 Network and load balancing configuration example

Table 2-3 defines the load balancer interfaces:

Table 2-3 Load balancer interfaces

Interface	IP address
Barracuda admin console	?http://10.45.3.70:8000
CoAP	[2001:470:1f15:16ea:20c:29ff:fe16:caca]:5683
CoAP eDTLS	[2001:470:1f15:16ea:20c:29ff:fe16:caca]:5684
NSP REST	http://10.45.3.70:8080

Barracuda configuration:

- Virtual IP: 2001:470:1f15:16ea::caca (External IP of CoAP).
- Virtual Port: 5683.
- Interface: WAN.
- Type: UDP Proxy.
- Advanced Options: Enable Client Impersonation: yes.
- Advanced Options: Timeout: 0.
- Persistence: Timeout: 0.

- Persistence: Client IP Port (having persistence on CoAP is not mandatory).
- Service Monitor: UDP Port check.
- Real Server 1 (NSP1):
  - IP: 2002:470:1f15:16ea::1001
  - Port: 5683.
- Real Server 2 (NSP 2):
  - IP: 2002:470:1f15:16ea::1002
  - Port: 5683.

Make sure that the `Advanced: Advanced Networking: IP Masquerading: Enable IP Masquerading for Real Servers = no`.

The load balancer must be transparent for the NSP and the connecting clients. This means that the NSP sees that the connecting client is coming from a correct IP address, and vice versa. To accomplish this, you can use client impersonation and the Barracuda as a gateway to the nodes. For more information on client impersonation, see https://techlib.barracuda.com/display/blbv42/client+impersonation?.

Barracuda IP configuration:

- WAN:
  - IP: 10.45.3.69
  - Mask: 255.255.255.0
  - Gateway: 10.45.3.1
  - IPv6/Mask: 2001:470:1f15:16ea::caca/64
- LAN:
  - LAN IP: 192.168.0.1
  - Mask: 255.255.255.0
  - IPv6/Mask: 2002:470:1f15:16ea::1/64

Table 2-4 defines the services available:

Table 2-4 Services

Name	IP	Type
CoAP	2001:470:1f15:16ea:20c:29ff:fe16:caca	UDP Proxy
NSP-1	2002:470:1f15:16ea::1001	
NSP-2	2002:470:1f15:16ea::1002	
REST API	192.168.0.70	TCP Proxy
NSP-1	192.168.0.41	
NSP-2	192.168.0.42	

NSP server configuration:

- In the NSP, the Barracuda server must be set as a gateway for outgoing messages (for example, route -A inet6 add 2001::/3 gw 2002:470:1f15:16ea::1). The NSP must not have a direct connection to 2001 network.
- Add the NSP REST network setup as follows:
  - NSP 1:

    ```
    [root@twix-nsp-1 ~]# ifconfig eth2 add 192.168.0.41
    [root@twix-nsp-1 ~]# ifconfig eth2:0 netmask 255.255.255.0
    [root@twix-nsp-1 ~]# route add 192.168.0.41 gw 192.168.0.1 eth2:0
    ```
  - NSP 2:

    ```
    [root@twix-nsp-2 ~]# ifconfig eth1 add 192.168.0.42
    [root@twix-nsp-2 ~]# ifconfig eth1:0 netmask 255.255.255.0
    [root@twix-nsp-2 ~]# route add 192.168.0.42 gw 192.168.0.1 eth1:0
    ```

The output of network configurations in NSP 1:

```
[root@twix-nsp-1 log]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:AC:E4:06
          inet addr:10.45.3.41  Bcast:10.45.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:910870 errors:0 dropped:0 overruns:0 frame:0
          TX packets:833861 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:395359636 (377.0 MiB)  TX bytes:250681140 (239.0 MiB)

eth2      Link encap:Ethernet  HWaddr 00:0C:29:AC:E4:10
          inet6 addr: 2002:470:1f15:16ea::1001/64 Scope:Global
          inet6 addr: fe80::20c:29ff:feac:e410/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101124 errors:0 dropped:0 overruns:0 frame:0
          TX packets:80867 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8919214 (8.5 MiB)  TX bytes:6483069 (6.1 MiB)

eth2:0    Link encap:Ethernet  HWaddr 00:0C:29:AC:E4:10
          inet addr:192.168.0.41  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:101 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:86095 (84.0 KiB)  TX bytes:86095 (84.0 KiB)

[root@twix-nsp-1 log]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.41    192.168.0.1     255.255.255.255 UGH   0      0        0 eth2
10.45.3.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth2
0.0.0.0         10.45.3.1       0.0.0.0         UG    0      0        0 eth0
[root@twix-nsp-1 log]#

[root@twix-nsp-1 log]# route -A inet6 -n
Kernel IPv6 routing table
Destination                                 Next Hop                                Flags Metric Ref    Use Iface
2002:470:1f15:16ea::/64                     ::                                      U     256    1        0 eth2
```

```
2000::/3                                    2002:470:1f15:16ea::1                   UG    1      61348       0
eth2
fe80::/64                                   ::                                      U     256    0        0 eth2
::/0                                        2002:470:1f15:16ea::1                   UG    1      7        0 eth2
::1/128                                     ::                                      U     0      9        1 lo
2002:470:1f15:16ea::1001/128                ::                                      U     0      78678       1 lo
fe80::20c:29ff:feac:e410/128                ::                                      U     0      5786       1 lo
ff00::/8                                    ::                                      U     256    0        0 eth2
[root@twix-nsp-1 log]#
```

2.2.4 Configuration files

You can find the NSP configuration files from the NSP installation package in the `/conf` directory. The `nsp.properties` file contains all configurations.

Each parameter is commented in the file.

Logging

By default, the NSP log is written to rolling logs in the `/log` directory and the log level configuration is read from the `log4j.properties` file in the `/conf` directory.

Event logger

Event logger stores communication events into a `csv` file in the `/log` directory. You can import it into Excel for analysis. Filtering the communication events can be configured in the `log4j.properties`.

The following event channels and default log level are used:

```
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.coap=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.edtls=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.register=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.update=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.unregister=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.proxy=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.eventing=INFO
log4j.logger.com.sensinode.nanoservice.eventlog.NanoEvent.m2mhttp=INFO
```

Failure events are logged at `WARN` and `ERROR` levels, normal communication events are logged as `INFO`.

Table 2-5 lists the event types used.

Table 2-5 Event types

Type	Description	Example details
Coap received	Incoming CoAP message	`CON POST MID:44386 URI:/rd?ep=node001 pl(37):0x3c2f70617468...`
Coap sent	Outgoing CoAP message	`ACK 201 MID:44386 Loc:/rd/domain/node001 MaxAge:3600s`
eDTLS session activated	Handshake succeeded and session is active	`Cipher: ECDSA`
eDTLS session terminated	Session was removed beccause of an error	`Alert [FATAL: DECRYPT_ERROR]`
eDTLS handshake failed	Failure while handshake message exchange	`Alert [FATAL: UNEXPECTED_MESSAGE]`

Table 2-5 Event types (continued)

Type	Description	Example details
eDTLS repeated	Received message was detected as repeated	`discarded`
EP registered	Endpoint has been registered	`name:stub-001, domain:test.domain.com, code:201, type:stub, lifetime:3600, binding:U`
EP registration failure	Endpoint has failed to register	`name:endpoint-1, domain:non-existing, code:404, message:Domain not found.`
EP updated	Endpoint registration has been updated	`name:stub-001, domain:test.domain.com, code:204, type:stub, lifetime:3600`
EP update failure	Endpoint has failed to update its registration	`name:endpoint-1, domain:non-existing, code:404, message:Endpoint not found.`
EP unregistered	Endpoint has been unregistered	`name:endpoint-1, domain:domain`
EP unregistration failure	Endoint has failed to unregister	`name:endpoint-1, domain:non-existing, code:404, message:Endpoint not found.`
Notifications delivered	Notification to client application has been delivered	-
Notifications delivery failed	Notification to client application failed	-
M2MHttp sent	Outgoing M2M Http message	`GET http://myhost:61221/path1 "Content-Type="...`
M2MHttp received	Incoming M2M Http message	`200 OK "Content-Length=7"; payload=0x5061736b610d0a`
Proxy request	Proxy request received from the web application	`ClientID=test.domain.com/testApp1, READ, node-001@test.domain.com/path1, sync=false, cacheOnly=false`
Removed endpoint	Endpoint was removed through admin	`endpoint-1@domain.com`

Endpoint templates

With endpoint templates you can create endpoint resource profiles that are connected to an endpoint type. If the endpoint type is configured in a server template, the server has preconfigured knowledge of the resources the endpoint has. The endpoint can register with the service simply by providing the endpoint type in the M2M endpoint registration.

An endpoint can contain the elements listed in Table 2-6.

Table 2-6 Endpoint elements

Element	Description
endpoint-type	Endpoint type that matches the endpoint template
resource	Preconfigured resource for this endpoint type

Resources can contain the elements listed in Table 2-7.

Table 2-7 Resource elements

Element	Description
resource-id	Individual resource identifier for this resource within the context of the same endpoint type
path	Resource path for the resource
type	Preconfigured resource type
content-type	Resource content type, for example "`text/plain`"
observable	Observable information. Can contain the values: `auto` auto-observable resource; `normal` observable resource; `no` non-observable resource

```
{
   "endpoint":[
      {
         "endpoint-type":"LocationSensor",
         "resource":[
            {
               "resource-id":"90",
               "path":"/gps/loc",
               "type":"ns:gpsloc",
               "observable":"auto"
            },
            {
               "resource-id":"93",
               "path":"/fw/devtype",
               "content-type":"text/plain",
               "type":"ns:devtype",
               "observable":"auto"
            },
            {
               "resource-id":"94",
               "path":"/fw",
               "content-type":"application/nanoservice-tlv",
               "type":"ns:root",
               "observable":"auto"
            }
         ]
      }
   ]
}
```

The templates are configured as json files.

Update trigger mechanism

When an endpoint has registered with the binding parameter `UQS`, the NSP can trigger the endpoint and force it online by sending an Update Trigger as an SMS. Endpoints connected using cellular do not require a continously active data connection. The data connection is only activated when it receives the SMS trigger.

To enable this feature, configure an SMS plugin in the `nsp.properties` file.

Figure 2-5 shows an example of how the NSP triggers an endpoint in Queue Mode to send an Update message regardless of the lifetime expiration.

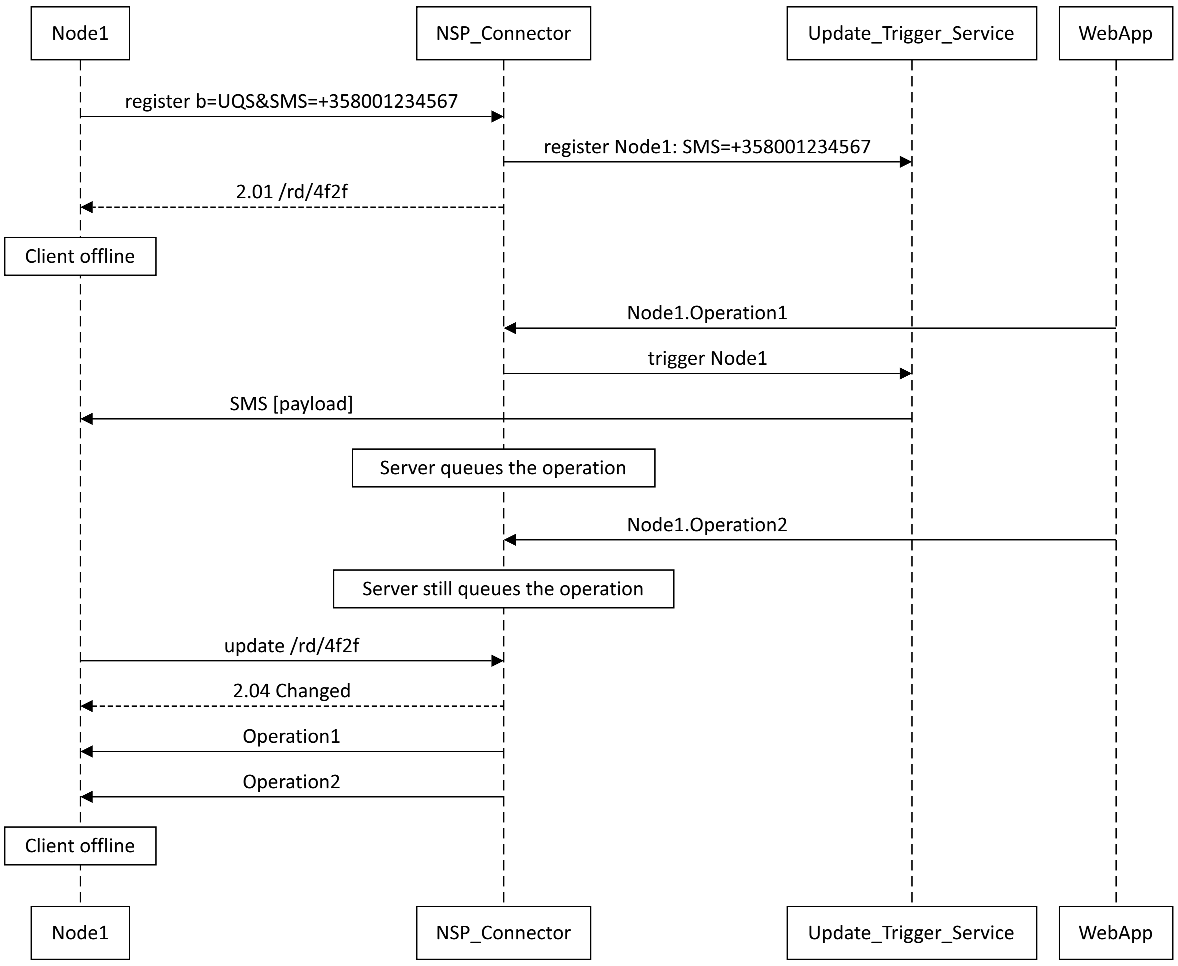


Figure 2-5 NSP triggers an endpoint

2.3 Secured SSL REST interface

For security reasons, it is possible to use secure HTTPS connections between the NSP and a connecting client using a REST interface.

Note

Java7 does not support all algorithms, such as MD2 (which has proven to be insecure).

Java keystore files are used for storing certificates and trusted public keys. The keystore file can be manipulated using a Java keytool command. For more details, see:

`http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`

The pre-configured example of keystores and configurations should not be used in product environment, it is only for testing secure connections.

To start the NSP in secure mode, follow these instructions:

1. Start the NSP in HTTPS mode:

   ```
   com.sensinode.server.https.in.use=true
   ```

2. Configure an NSP keystore file, for example:

   ```
   com.sensinode.server.https.keystore.file=../conf/nsp.keystore
   ```

3. Configure a password for the NSP keystore file in obfuscated format:

   ```
   com.sensinode.server.https.keystore.password=!
   OBF:1tvd19q71v231ktv1ktz1l161kxw1kqv1kqz1v2b19q71tvp
   ```

   You can obfuscate a clear text password by running `org.mortbay.jetty.security.Password` as a main class.

   For more information on obfuscation, see the *Password issues* in `http://wiki.eclipse.org/Jetty/Howto/Configure_SSL`.

If the used certificate is self signed and the connecting client requires a public key to accept it, you can use Java keytool to export the public key required.

If the connecting client uses a secured subscription server for receiving notifications, the NSP must accept the client server certificate to allow connection. This can be achieved in two ways:

- Have a signed certificate in the client subscription server that can be found in a default, trusted certificate storage in the operating system where the NSP is running.
- Configure a used public key to a trusted keystore:
  - Configure an NSP trusted keystore file, for example:
    `'com.sensinode.server.https.trusted.keystore.file=../conf/trust.keystore'`
    This trusted keystore must contain the public key of the client certificate.
  - Configure a password for the NSP trusted keystore as clear text:
    `'com.sensinode.server.https.trusted.keystore.password=d4hjln346l4j'`.

2.4 Keystore creation with the keytool command

You can manipulate keystore files with the `keytool` executable shipped in with Java JRE/SDK. For more information, see `http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`.

The following commands are available:

- To create a new NSP certificate to a new keystore, give the following command:

  ```
  keytool -genkeypair -alias myalias -keypass mypassword -
  keystore mykeystore.keystore
  ```

  This produces a keystore file for the NSP.

  You can get an obfuscated password of the keystore password with `org.mortbay.jetty.security.Password` described in `http://wiki.eclipse.org/Jetty/Howto/Configure_SSL`.

  This produces an obfuscated password for the keystore.

- To export a certificate from the keystore, give the following command:

  ```
  keytool -export -alias myalias -file mycert.crt -keystore mykeystore.keystore
  ```

- To create a new keystore for a trusted certificate, give the following command:

  ```
  keytool -import -trustcacerts -alias myalias -file mycert.crt
   - keystore mytrust.keystore
  ```

  This produces a trusted keystore file for the client's subscription server connection. Password given during the keystore creation is the field for a trusted keystore password.

2.5 eDTLS configuration

The NSP eDTLS supports authentication with a Pre-Shared Key and Certificates.

2.5.1 PSK credentials

To set the initial eDTLS Pre-Shared Key credentials to be loaded on the NSP startup, use the properties:

```
com.sensinode.server.coap.edtls.pskfile={filename}
```

The credentials file format is *Comma Separated Variables* (CSV) that you can edit with Excel. The following is an example credentials file:

```
##
# Format:
# EndPointName (optional), KeyID(HEX), Bootstrap key ID(HEX, optional), Auth (boolean),
Registration(boolean), Network-join(boolean), Secret (HEX)
# boolean: 0 (true) | 1 (false)
#

,0F0F,,0,1,1,00FF
SomeName,0F10,00FF,0,1,1,FFFFAAAA332211
,0F11,CACA,0,1,1,00
something,ABCD,,1,1,1,00
```

2.5.2 Certificate

You can load the initial Certificates on the NSP startup with properties:

```
com.sensinode.server.coap.edtls.certfile=../conf/nanoservice.bks
com.sensinode.server.coap.edtls.certfile.password=testpassword
com.sensinode.server.coap.edtls.cert.alias=nsp-signed
com.sensinode.server.coap.edtls.cert.privatekeyalias=nsp
com.sensinode.server.coap.edtls.cert.ca.alias=ca
```

Create the keystore file nanoservice.bks with the NanoService Key Tool (default command `create-keystore.sh`). The default values that the tool uses are seen on the properties above.

You can edit the Certificates later in the AdminUI Security page. When updating the Certificate information, the format of the new certificate data, and key files that can be exported from the keystore with the NanoService Key Tool (default commands `export-cert.sh ca`, `export-cert.sh nsp-signed` and `export-key.sh nsp`), is plain text base64 output of pem. Update the data by pasting the certificate data to the text area.

Example:

```
-----BEGIN CERTIFICATE-----
MIIBfzCCASOgAwIBAgIEc+6p0jAMBggqhkjOPQQDAgUAMDQxCzAJBgNVBAYTAkZJ
MRQwEgYDVQQKEwtBUk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMB4XDTE0MDEw
MjEyNDA1OVoXDTIzMTIzMTEyNDA1OVowNDELMAkGA1UEBhMCRkkxFDASBgNVBAoT
C0FSTSBGaW5sYW5kMQ8wDQYDVQQDEwZSb290Q0EwWTATBgcqhkjOPQIBBggqhkjO
PQMBBwNCAATQGo//vP80an+ntsGT7B1Cjj8ubofz1h+mlHntxJE/et/4+xU29ooE
TYjbvIv8XewVdJ1QsEb48JfiD0+dQZVvoyEwHzAdBgNVHQ4EFgQUpPPiV0sB/rw6
0BMmIIBdAeztuJ8wDAYIKoZIzj0EAwIFAANIADBFAiEAlGhSUTBIT8HTxZB8r2ur
8G7v+ZWM9wWoYXrRAootWZICIFSzzpbTr3RVHBcZF0hDWZ04eWmb4tJWG2z5t6+s
3Gu+
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBRjCB7aADAgECAgRAJhsWMAoGCCqGSM49BAMCMDQxCzAJBgNVBAYTAkZJMRQw
EgYDVQQKEwtBUk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMB4XDTE0MDEwMjEy
NDEwMloXDTE1MDEwMjEyNDEwMlowIzELMAkGA1UEBhMCRkkxFDASBgNVBAoTC0FS
```

```
TSBGaW5sYW5kMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEt7KwoDD5C7h0KAKn
2CgbTIj2Kz9y8rtjfOfr/xgrtNMpdLguKuVZIg3ijrsfVTsD2256AfV6W7om+BFv
NqD7MDAKBggqhkjOPQQDAgNIADBFAiAizSCRXLSNyXfUtfJxE/drsWzhZLRdg4F1
QhhJ0DXA3gIhANemcIesJJ2vo0+LwY6jWYp2BmUbDpde1oFqIqAVTcvH
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
ME0CAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQcEMzAxAgEBBCDKI1pWcKAz6dwYuQPA
+OkCgiU0WLlQ8H0jxU51G5tFwKAKBggqhkjOPQMBBw==
-----END PRIVATE KEY-----
```

Chapter 3
M2M Interfaces

This chapter describes the M2M interfaces of the NanoService Platform. It contains the following sections:

3.1 Introduction

The NanoService Platform (NSP) provides Machine-to-Machine (M2M) interfaces through which M2M nodes interact with the NSP. The interfaces are available over the CoAP protocol [draft-ietf-core-coap-18], and over HTTP (M2M-HTTP) which is disabled by default. Observe and notification messages are not supported over HTTP.

For easy integration of a device with the NanoService Platform, use the appropriate NanoService Device Library (Java and C).

3.2 NanoService platform data model

The NSP data model is divided into:

- Resources.
- Endpoints.
- Groups.
- Domains.

Resources can be perceived as individual URI paths, for example, `/path, /longer/path`. Several resources can be assigned to a single endpoint. Resources are typically used to provide access to sensors, actuators and configuration parameters on an M2M device. An endpoint is the web server software running on a device and it belongs to some domain.

In short, a domain contains endpoints that contain resources. Several domains can be configured in the NanoService Platform.

NanoService endpoints can also be grouped into logical groups. Each endpoint can belong to several groups.

NanoServices allows endpoints and resources to be associated with semantic naming. This means that during registration, naming metadata can be associated with them. An endpoint includes a host name that uniquely identifies it in a domain. An endpoint can also be associated with a node type, for example “MotionDetector”. Resources can be associated with a resource type (for example “LightSensor”), an interface description, and other metadata about the resource, such as its content types and observability.

3.3 M2M server interfaces

Table 3-1 lists the components available through the NSP M2M interfaces. The M2M interfaces are accessed using CoAP.

Table 3-1 NSP components

NSP component	URI path	Note
Resource discovery	`/.well-known/core`	-
Resource directory	`/rd`	-
Authentication	`/auth`	Available only in secure connection (eDTLS).

Note

The default port for non-encrypted communication is 5683, and for encrypted, 5684. You can configure the ports in the `nsp.properties` file.

3.4 Resource discovery

The well-known CoRE resource includes the M2M interfaces that the NSP offers under `/.well-known/core` with the REST interface defined in ?[RFC 6690]. An M2M node uses it to discover the NSP's interface automatically.

——— **Note** ———

This interface does not support query filtering.

———————

To retrieve the NSP resources, use this method:

```
GET /.well-known/core
```

The `/.well-known/core` resource contains the following link-format descriptions:

```
</rd>;rt="core-rd"
```

3.5 Resource directory

The resource directory enables the registration of endpoints with the NanoService Platform. An endpoint is registered by making a request to `/rd`. The request query string may contain any of the endpoint attributes defined in Table 3-2, and the request body is a list of the resources to register for that endpoint in the Link Format. Each link may contain any of the resource attributes defined in Table 3-2.

A registration with the resource directory has a soft state, and must be refreshed periodically. A non-default lifetime can be specified by including the lifetime parameter in the request. All the resources that are previously stored in the resource directory but are not delivered in the request body, are removed, and all the new resources are added to the resource directory.

3.5.1 Registration

To register an endpoint, use this method:

```
POST /rd?ep={endpoint-name}&et={endpoint-type}&lt={lifetime}&con={context-address}
&d={domain}&b={binding}
```

This request registers an endpoint with the NSP with the specified query parameters and a list of resources formatted in the Link Format [?RFC 6690] as the payload of the message.

Table 3-2 lists the attributes supported.

Table 3-2 Supported attributes

Attribute	Name	Restrictions
`url`	Resource path	-
`rt`	Resource type	Only once for registration
`if`	Interface description	Only once
`ct`	CoAP content type	Convert ASCII MIME type
`obs`	Observable	Boolean
`aobs`	Auto-observable	Boolean
`id`	Resource identifier (used in auto-observation)	Integer

Table 3-3 lists the query parameters supported.

Table 3-3 Supported query parameters

Query parameter	Name	Restrictions
`ep`	Endpoint name. A unique name for the registering node in a domain.	(Optional) 63 bytes. If the parameter is not present in the request, the name is generated. The name has to be [RFC1123] compliant, with the exclusion that dots(.) are not allowed in a host name.
`et`	Endpoint type	(Optional) 63 bytes
`lt`	Lifetime. Number of seconds that this registration will be valid for. Must be updated within this time, or will be removed.	(Optional) 32-bit uint

Table 3-3 Supported query parameters (continued)

Query parameter	Name	Restrictions
`d`	Domain name	(Optional) 63 bytes. If this parameter is missing, a default domain is assumed.
`con`	Address at which this endpoint is available. In the absence of this parameter, the scheme of the protocol, a source address of the register request is assumed. This parameter is mandatory for M2M-HTTP binding.	(Optional for CoAP) scheme://host:port
`b`	Indicates the current binding mode and queue mode of the registering device: `U` UDP. `Q` Queue mode. `S` SMS binding. On M2M-HTTP interface, this parameter is not supported.	(Optional) SMS binding is not supported.

The following deprecated query parameters are still supported:

`h`	Endpoint name.
`rt`	Endpoint type.

Table 3-4 lists the response codes.

Table 3-4 Response codes

CoAP code	Description
`2.01 Created`	Successful registration, returns the URI location that is used in updated registration and max-age.
`4.00 Bad request`	Malformed message: • Required query parameter is missing. • Template is not found. • Domain does not exist.

Example:

```
POST /rd?h=node-001&rt=Light&lt=6000
Uri-Host: domain.com

</light>;rt="ucum:Lux";if="ns.wadl#s",</uv-light>;rt="ns:image";if="ns.wadl#s"
=>
2.01 Created
Location-Path: /rd/domain.com/node-001
```

Templates

Registrations can be done by using endpoint templates described in the configuration guide. Templates allow predefined resource sets to be used for a registered endpoint. A template is mapped by an endpoint type.

3.5.2 Update registration

Endpoint information can be updated by sending a `PUT` request to `/rd/{reg-location-path}`. Resources can also be added by including a body in the request. The resources delivered in the message body are handled as new resources for the endpoint.

To update endpoint information, use this method:

```
PUT /rd/{domain}/{name}?lt={lifetime}
```

Example:

```
PUT /rd/domain.com/node-001?lt=6000
=>
2.04 Changed
```

M2M-HTTP

On the HTTP interface, the `con` parameter is mandatory for update registration.

3.5.3 De-registration

Endpoint information can be removed by sending a `DELETE` request to `/rd/{reg-location-path}`. The message does not contain a body and all the resources associated with the deleted endpoint are removed from the resource directory.

To remove endpoint information, use this method:

```
DELETE /rd/{domain}/{name}
```

Example:

```
DELETE /rd/domain.com/node-001
=>
2.02 Deleted
```

3.6 Authentication interface

With authentication functionality `/auth`, the PANA server can request the eDTLS pre-shared secret of any endpoint. This is enabled only via eDTLS secured UDP connection. The PANA server must be defined in the `whitelist.csv` with authentication enabled. A successful response contains the secret of the requested `key-id` in hex format.

```
GET /auth?pskid={pre-shared-key-id (HEX)}
accept: 42 | 0 (APPLICATION/OCTET-STREAM | TEXT/PLAIN)
```

Table 3-5 lists the response codes.

Table 3-5 Response codes

CoAP code	Description
`2.05 Content`	Returns a valid secret (as raw bytes or HEX).
`4.01 Unauthorized`	Key-id could not be found in the white list.
`4.03 Forbidden`	Requester does not have permission to access this interface.

3.7 Batch notifications

The NanoService binary TLV format is used to package a batch of resource representations in a single payload. This format can be used for the NanoService root resource (`/`) or for a function set resource (for example, `/sen`). This format can be used as the payload in notifications, or in the response to a `GET` request.

Table 3-6 shows the media type supported.

Table 3-6 Supported media type

Media type	CoAP code
`application/nanoservice-tlv`	`200`

In this format, the ID field corresponds to the `id=` integer attribute assigned to the resource by the node. This same `id=` attribute is used for auto-observation. The binary format is simply the following structure repeated for each representation in the payload. The overall length of the payload is calculated from the IP packet length, therefore there is no length field.

Table 3-7 describes the TLV element fields.

Table 3-7 TLV element fields

Field	Format and length	Description
ID	16-bit unsigned integer	The NanoService `id=` link attribute corresponding to the resource
Length	16-bit unsigned integer	The length of the following field in bytes
Value	Sequence of bytes, length indicated by the Length field	Either a plain text or opaque value depending on the data format of the resource

Table 3-8 lists the ID values reserved.

Table 3-8 Reserved ID values

ID	Name	Description
0	Current timestamp	MUST be a Unix timestamp in seconds since 1970. Value represented as a string.
1	Time offset	The time in seconds before the current timestamp that the following measurements were made. Value represented as a string.

3.7.1 TLV example

In this example a node has two resources and the following link description:

```
</sen>;aobs;id="20";ct="200",
</sen/temp>;aobs;id="25",
</sen/light>;aobs;id="26"
```

When the node sends a notification to the server for `/sen`, or responds to a `GET` on `/sen`, when `temp=22.3` and `light=100`, the payload would be as follows.

3.7.2 Notifications with node's timestamps

To send a resource notification with a timestamp, the TLV batch message must contain the current timestamp and optionally an offset. Every following resource representation has a measurement time of the current timestamp minus offset. It is possible to have multiple time series by including a multiple timest offset.

Note

The current timestamp must always appear first and one time only.

Table 3-9 lists the node's timestamps.

Table 3-9 Node's timestamps

Byte #	Value	HEX	Note
0	0	`00`	id=25
1	25	`19`	-
2	0	`00`	length=4
3	4	`04`	-
4	'2'	`32`	value=22.3
5	'2'	`32`	-
6	'.'	`2e`	-
7	'3'	`33`	-
8	0	`00`	id=26
9	26	`1a`	-
10	0	`00`	length=3
11	3	`03`	-
12	'1'	`31`	value=100
13	'0'	`30`	-
14	'0'	`30`	-

Table 3-10 shows an example with the latest measurements.

Table 3-10 Example with latest measurements

ID	Value	Note	TLV raw data (HEX)
0	1357056000	2013.01.01 10:00	ID LEN VALUE `0000 000A 3133 3537 30353 6303 030`
25	20.1	temperature value=20.1 timestamp='2013.01.01 10:00'	`0019 0004 3230 2e31`
26	98	light value=20.1 timestamp='2013.01.01 10:00'	`001A 0004 3938`

Table 3-11 shows an example with historical measurements.

Table 3-11 Example with historical measurements

ID	Value	Note	TLV raw data (HEX)
0	1357056000	2013.01.01 10:00	ID LEN VALUE 0000 000A 3133 3537 3035 3630 3030
1	300	5 minutes offset	`0001 0003 3330 30`
25	20.1	temperature value=20.1 timestamp='2013.01.01 9:55'	`0019 0004 3230 2e31`
26	98	light value=20.1 timestamp='2013.01.01 9:55'	`001A 0004 3938`
1	600	10 minutes offset	`0001 0003 3630 30`
25	22.0	temperature value=22.0 timestamp='2013.01.01 9:50'	`0019 0004 3232 2e30`

Chapter 4
Web Interfaces

This chapter describes the web interfaces of the NanoService Platform. It contains the following sections:

4.1 Introduction

Web applications interact with the NanoService Platform using a set of RESTful Web interfaces over HTTP. These interfaces provide administration, lookup and eventing services.

4.2 NanoService platform data model

The NSP data model is divided into:

- Resources.
- Endpoints.
- Groups.
- Domains.

Resources can be perceived as individual URI paths, for example, `/path, /longer/path`. Several resources can be assigned to a single endpoint. Resources are typically used to provide access to sensors, actuators and configuration parameters on an M2M device. An endpoint is the web server software running on a device and it belongs to some domain.

In short, a domain contains endpoints that contain resources. Several domains can be configured on the NanoService Platform.

NanoService endpoints can also be grouped into logical groups. Each endpoint can belong to several groups.

NanoServices allows endpoints and resources to be associated with semantic naming. This means that during registration, naming metadata can be associated with them. An endpoint includes a host name that uniquely identifies it in a domain. An endpoint can also be associated with a node type, for example 'MotionDetector'. Resources can be associated with a resource type (for example 'LightSensor'), an interface description, and other metadata about the resource, such as its content types and observability.

4.3 Authentication

The security model used in the NSP is domain level based. Client applications are assigned to one domain. The credential configuration file `conf/credentials.properties` contains initial accounts that are loaded into cache.

Authorization is performed with HTTP basic authentication and every request must include an HTTP authorization header.

```
Authorization: Basic {base64encode(username:password)}
```

Every request contains a domain name in the first path segment:

```
/{domain-name}/...
```

4.4 Endpoint directory lookups

The directory lookup interface provides the possibility to browse and filter through the resource directory of the NSP.

4.4.1 List all endpoints

To list all endpoints, use the method provided in Table 4-1.

Table 4-1 List all endpoints

Name	Description	
URI	`/{domain}/endpoints?type={endpoint-type}&stale={true|false}`	
Method	GET	
Query parameters	`type`	Filters endpoints by endpoint-type. Optional.
	`stale`	If `true`, the result list will contain stale endpoints. Optional. The default is `false`.
Responses	`200`	Successful response with a list of endpoints
Acceptable content-types	application/json application/link-format	

Endpoint list `json` structure:

```
[
    {   "name": "{endpoint name}",
        "type": "{endpoint type}",
        "status": "{endpoint status: ACTIVE|STALE}"
        "q": "{queue mode: true|false (default: false)"}
]
```

Example:

```
GET /example.com/endpoints
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

[
    { "name":"node-001", "type":"Light", "status":"ACTIVE"},
    { "name":"node-002", "type":"Light", "status":"STALE"},
    { "name":"node-003", "type":"QueueModeNode", "status":"ACTIVE", "q": true}
]
```

4.4.2 List endpoint's resources metainformation

To list the metainformation of the endpoint's resources, use the method provided in Table 4-2.

Table 4-2 List endpoint's resources metainformation

Name	Description	
URI	`/{domain}/endpoints/{endpoint-name}`	
Method	GET	

Responses	`200`
	`404`
		OK
		Endpoint not found

Acceptable content-types	application/json application/link-format	

Example:

```
GET /example.com/endpoints/node-001
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

[
    { "uri":"/dev/temp", "rt":"ucum:C", "obs":"true", "type":"text/plain"},
    { "uri":"/dev/illu", "obs":"false", "type":"text/plain"}
]
```

4.4.3 Endpoint's resource representation

To access the endpoint's resource representation, use the method provided in Table 4-3.

Table 4-3 Endpoint's resource representation

Name	Description
URI	`/{domain}/endpoints/{endpoint-name}/{resource-path}?sync={true|false}&cacheOnly={true|false}&pri={dscp}&noResp={true|false}`
Method	GET, PUT, POST, DELETE

Table 4-3 Endpoint's resource representation (continued)

Name	Description	
Query parameters	sync	Indicates whether this request is synchronous or asynchronous. Optional. The default is `false`.
	cacheOnly	If `true`, the response will only come from cache. Optional. The default is `false`.
	pri	Priority message. Adds traffic-class for outgoing IPv6 message (only UDP). Accepted values are: `AF11`, `AF12`, `AF13`, `AF21`, `AF22`, `AF23`, `AF31`, `AF32`, `AF33`, `AF41`, `AF42`, `AF43`, `VA`, `EF`, `CS0`, `CS1`, `CS2`, `CS3`, `CS4`, `CS5`, `CS6`, `CS7` and `DF`. Numeric values `0-7` are interpreted as matching to the corresponding CS value. Optional. The default is `0`.
	noResp	If `true`, no waiting for response and no response is expected. Creates CoAP Non-Confirmable requests. Optional. The default is `false`.
Responses	200	Successful GET, PUT, DELETE operation
	201	Successful POST operation
	202	Accepted. Asynchronous response ID (only when `sync=false`).
	205	No cache available for resource
	404	Requested endpoint's resource is not found
	408	Endpoint did not respond, time-out (only when `sync=true`)
	409	Conflict. Endpoint is in queue mode and synchronous request cannot be made (only when `sync=true`), or if `noResp=true`, then means that the request is not supported.
	410	Gone. Endpoint not found.
	429	Cannot make a request at the moment, already ongoing other request for this endpoint
Acceptable content-types	application/json (for `async-response-id`)	
	/ (depends on the endpoint's resource)	

Asynchronous

Activated when the `sync` parameter is missing or false. The endpoint's response will arrive in the notification channel. An HTTP request will return immediately with `async-response-id` that is used to match the response. For GET requests, if the representation is available in the cache, it will return it.

Example:

```
GET /example.com/endpoints/node-001/dev/temp

HTTP/1.1 202 Accepted
Content-Type: application/json

{"async-response-id": "23471#node-001@test.domain.com/path1"}
```

Synchronous (not recommended)

Activated when the `sync` parameter is true. A request will hold until a response comes, or a time-out. Requests for endpoints in queue mode are not supported.

Example:

```
GET /example.com/endpoints/node-001/dev/temp?sync=true

HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: max-age=100

24.5C
```

No response (NON)

Indicated with the parameter `noResp=true` and as a result, the NSP makes CoAP Non confirmable requests. The REST response is with the status code `204 No Content`. If the underlying protocol does not support it (HTTP) or the endpoint is registered in queue mode, the response is with the status code `409 Conflict`.

Note

These request types are not guaranteed.

Example:

```
GET /example.com/endpoints/node-001/dev/temp?noResp=true
```

```
HTTP/1.1 204 No Content
```

4.5 Notifications

The NanoService eventing model consists of observable resources, which enables endpoints to deliver updated resource content, periodically or with a more sophisticated solution-dependent logic. Applications must subscribe to every individual resource to receive a notification update.

4.5.1 Subscribe to a resource

To subscribe to a resource, use the method provided in Table 4-4.

Table 4-4 Subscribe to a resource

Name	Description	
URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}?sync={true|false}`	
Method	PUT	
Query parameters	`sync`	Indicates whether this subscription request is synchronous or asynchronous. Optional. The default is `false`.
Responses	`200`	Successfully subscribed
	`202`	Accepted. Asynchronous response ID (only when `sync=false`).
	`404`	Endpoint or endpoint's resource not found
	`412`	Cannot make a subscription for a non-observable resource
	`429`	Cannot make a subscription request at the moment due to an already ongoing other request for this endpoint
	`502`	Subscription failed
	`504`	Subscription could not be established due to a time-out from the endpoint

Asynchronous

Activated when the `sync` parameter is missing or false. A subscription response will arrive in the notification channel. An HTTP request will return immediately with `async-response-id` that is used to match the response. If the subscription can be established immediately, without contacting the endpoint, this call will return the subscription result.

Example:

```
PUT /example.com/endpoints/node-001/dev/temp

HTTP/1.1 202 Accepted
Content-Type: application/json

{"async-response-id": "5734979#node-001@test.domain.com/path1"}
```

4.5.2 Remove a subscription

To remove a subscription, use the method provided in Table 4-5.

Table 4-5 Remove a subscription

Name	Description	
URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}`	
Method	DELETE	
Responses	`200`	Successfully removed subscription
	`404`	Endpoint or endpoint's resource not found

4.5.3 Read subscription status

To read subscription status, use the method provided in Table 4-6.

Table 4-6 Read subscription status

Name	Description	
URI	`/{domain}/subscriptions/{endpoint-name}/{resource-path}`	
Method	GET	
Responses	`200`	Resource is subscribed
	`404`	Resource is not subscribed

4.5.4 Read endpoint's subscriptions

To read endpoint's subscriptions, use the method provided in Table 4-7.

Table 4-7 Read endpoint's subscriptions

Name	Description	
URI	`/{domain}/subscriptions/{endpoint-name}`	
Method	GET	
Responses	`200`	List of subscribed resources
	`404`	Endpoint not found
Acceptable content-types	text/uri-list	

Example:

```
GET /example.com/subscriptions/node-001

HTTP/1.1 200 OK
Content-Type: text/uri-list

/example.com/subscriptions/node-001/dev/temp
/example.com/subscriptions/node-001/dev/power
```

4.5.5 Remove endpoint's subscriptions

To remove endpoint's subscriptions, use the method provided in Table 4-8.

Table 4-8 Remove endpoint's subscriptions

Name	Description	
URI	`/{domain}/subscriptions/{endpoint-name}`	
Method	DELETE	
Responses	`200`	Successfully removed
	`404`	Endpoint not found

4.5.6 Receiving notifications

Notifications can be delivered with two different mechanisms:

- The "Long polling".
- Using a subscription server where the NSP actively sends the notifications.

The notifications are delivered in JSON format.

Client's push url

Notifications are delivered as PUT messages to the HTTP server defined by the client with a subscription server message. The client requires the following REST request (subscription):

```
PUT /{domain}/notification/push-url
```

The entity of the message must contain an URL location where the client listens to incoming notifications. If a special host name is used: `REMOTE_HOST`, the NSP will replace it with a source IP address.

Example:

```
PUT /example.com/notification/push-url
Content-length: 30

http://192.168.5.5:8089/events

HTTP/1.1 200 OK
```

The following request returns the client's URL:

```
GET /{domain}/notification/push-url
```

Long polling for notifications

Notifications are delivered through HTTP long-poll requests. The HTTP request is kept open until an event notification or a batch of event notifications is delivered to the client, or a request time-out occurs. In both cases, the client should open a new polling connection after the previous one closes. Long polling connections are handled on a per domain basis, so each open connection delivers notifications for a single domain. The interface for receiving event notifications has an URL of the form:

```
GET /{domain}/notification/pull
```

4.5.7 Notification data structure

A notification message contains the server event types provided in Table 4-9.

Table 4-9 Notification types

Notification type	Description
notifications	Contains resource notifications
registrations	List of endpoints that have registered (with resources)
reg-updates	List of endpoints that have updated registration
async-responses	Responses to asynchronous proxy request

The notification message is delivered in JSON format, as in the following example:

```
{
    "notifications":[
        {
            "ep":"{endpoint-name}",
            "dm":"{domain}",
            "path":"{uri-path}",
            "ct":"{content-type}",
            "payload":"{base64-encoded-payload}",
            "timestamp":"{timestamp}"
            "max-age":"{max-age}"
        }
    ],
    "registrations":[
        {
            "ep": "{endpoint-name}",
            "ept": "{endpoint-type}",
            "q": "{queue-mode, default: false}",
            "resources": [ {
                "path": "{uri-path}",
                "if": "{interface-description}",
                "rf": "{resource-type}"
                "ct": "{content-type}",
                "obs": "{is-observable (true|false) }"
            } ]
        }
    ],
    "reg-updates":[
        {
            "ep": "{endpoint-name}"
            "ept": "{endpoint-type}"
        }
    ],
    "async-responses":[
        {
            "id": "{async-response-id}",
            "status":  {http-status-code},
            "error":  {error-message},
            "ct":  "{content-type}",
            "max-age": {max-age},
            "payload":  "{base64-encoded-payload}"
        }
    ]
}
```

4.6 Groups

The NanoService endpoints can be grouped into logical groups. The following sections describe requests associated with groups.

4.6.1 List all root groups

To list all root groups, use the method provided in Table 4-10.

Table 4-10 List all root groups

Name	Description	
URI	`/{domain}/groups?all=(true|false)`	
Method	GET	
Query parameters	`all`	If `true`, the result will contain all groups, otherwise only root groups. Optional. The default is `false`.
Responses	`200`	Successful response
Acceptable content-types	application/json application/link-format	

Example:

```
GET /example.com/groups

HTTP/1.1 200 OK
Content-Type: application/json

["group-1", "group-2", "group-3"]
```

4.6.2 Read group content

To read the group content, use the method provided in Table 4-11.

Table 4-11 Read group content

Name	Description	
URI	`/{domain}/groups/{group-name}`	
Method	GET	

Responses	`200`
	`404`
		Successful response
		Group not found

Acceptable content-types	application/json	

Example:

```
GET /example.com/groups/group-1

HTTP/1.1 200 OK
Content-Type: application/json

{
    "name": "group-1"
    "description": "Group description"
    "endpoints: ["node-001", "node-002", "node-003"]
    "subGroups": ["group-101", "group-102"]
}
```

4.6.3 Update group content

To update group content, use the method provided in Table 4-12.

Table 4-12 Update group content

Name	Description	
URI	`/{domain}/groups/{group-name}?removeMembers={true|false}`	
Method	PUT	
Query parameters	`removeMembers`	If `true`, the given members will be removed from the group. Optional. The default is `false`.
Responses	`200`	Successful response
	`404`	Group not found
Acceptable content-types	application/json	

Example:

```
PUT /example.com/groups/group-1
Content-Type: application/json

{
    "name": "group-1",
    "description": "Group description",
    "endpoints: ["node-004"],
    "subGroups": ["group-104", "group-105"]
}

HTTP/1.1 204 No Content
```

4.6.4 Create a group

To create a group, use the method provided in Table 4-13.

Table 4-13 Create a group

Name	Description	
URI	`/{domain}/groups`	
Method	POST	
Responses	`202`	Group successfully created
Acceptable content-types	application/json	

Example:

```
POST /example.com/groups
Content-Type: application/json

{
    "name": "group-9",
    "description": "Group description",
    "endpoints: ["node-009"],
    "subGroups": ["group-109"]
}
```

```
HTTP/1.1 204 No Content
```

4.6.5 Remove a group

To remove a group, use the method provided in Table 4-14.

Table 4-14 Remove a group

Name	Description	
URI	`/{domain}/groups/{group-name}`	
Method	DELETE	
Responses	`204`	Group successfully removed
	`404`	Group not found

Example:

```
DELETE /example.com/groups/group-9
```

```
HTTP/1.1 204 No Content
```

4.7 XRI support

NSP rest interfaces support XRI lookups. Each response contains resolved links.

To resolve an endpoint by name, use the following method:

```
GET /{domain}/xri/*{endpoint-name}
```

To resolve root groups, use the following method:

```
GET /{domain}/xri/+group
```

To resolve group members, use the following method:

```
GET /{domain}/xri/+group:{group-name}
```

Example:

```
GET /example.com/xri/+group

HTTP/1.1 200 OK
Content-Type: text/url-list

/example.com/xri/+group:group1
/example.com/xri/+group:group2
/example.com/xri/+group:group3
```

4.8 NSP administration

Administration can be handled with an HTML user interface (**admin-ui**) or REST API (**admin-api)**. The interfaces can be accessed by default on ? https://localhost:8081, if not changed in `nsp.properties`.

The Admin web console can handle the following administration operations:

- Status:
  - Shows connector information, such as protocol, port and warnings.
- Endpoints:
  - Shows endpoints with filtering. Enables removing an endpoint.
- Security:
  - Manages eDTLS credentials.
- Clients:
  - Manages client application credentials.

The REST API can handle the following administration operations:

- Directory:
  - Retrieve, create and remove a domain.
  - Filter and remove endpoints.
- Security eDTLS PSK credentials:
  - List, filter, create, update and remove credentials.
  - Paging and sorting.
- Security eDTLS certificates:
  - Retrieve, remove and set certificate information.
- Client credentials:
  - Retrieve clients.
  - Create and update client credentials.
- Black list:
  - Retrieve blacklisted endpoints.
  - Add, remove and check endpoints in a black list.

4.8.1 Directory

To retrieve all domains, use the method provided in Table 4-15.

Table 4-15 Get all domains

Name	Description	
URI	`/admin-api/rd`	
Method	GET	
Responses	`200`	List of all domains

To create a new domain, use the method provided in Table 4-16.

Table 4-16 Create a new domain

Name	Description	
URI	`/admin-api/rd`	
Method	PUT	
Responses	`204`	Success
	`400`	Invalid domain name
	`400`	Domain already exists

To remove a domain, use the method provided in Table 4-17. The domain to be removed must have no endpoints.

Table 4-17 Delete a domain

Name	Description	
URI	`/admin-api/rd`	
Method	DELETE	
Responses	`204`	Success
	`400`	Domain already has endpoints
	`404`	Domain not found

To filter endpoints by name, type or status, use the following method:

```
GET
/admin-api/rd/{domain}?name={endpoint-name}&status={ACTIVE|STALE}&type={endpoint-type}&
maxEntries={max}
```

To remove an endpoint from the NSP, use the following method:

```
DELETE /admin-api/rd/{domain}/{endpoint-name}
```

4.8.2 Security eDTLS PSK credentials

To list all credentials, use the following method:

```
GET /admin-api/security/credentials
```

The list of all credentials contains the `originalKey` (to be used in UI when editing the actual key), `key` (HE), `secret` (Hex), `name`, authorization (bool), network (bool), and registration (bool).

Note

The `auth` and `nw` parameters are used for PANA server as described in *Authentication interface* on page 3-9. If the registration and authorization parameters are set to `false`, the endpoint eDTLS registration with the given `key-id` will be rejected.

Example return value:

```
[
    {"originalKey":"00","key":"00","auth":true,"nw":true,"reg":true,"name":"name",
    "secret":"07d0"},
    {"originalKey":"01","key":"01","auth":true,"nw":true,"reg":true,"name":"name",
    "secret":"07cf"},
    {"originalKey":"02","key":"02","auth":true,"nw":true,"reg":true,"name":"name",
    "secret":"07ce"},
    {"originalKey":"03","key":"03","auth":true,"nw":true,"reg":true,"name":"name",
    "secret":"07cd"}
]
```

To filter credentials by `key-id`, use the following method:

```
GET /admin-api/security/credentials?key={key-id (hex)}
```

`GET` supports paging with a default jqgrid syntax. When using paging, the return value also contains page information:

```
GET /admin-api/security/credentials?page={x}&rows={y}
```

Example:

```
{
    "total":10,"page":1,"records":100,
    "rows":[
        {"originalKey":"00","key":"00","auth":true,"nw":true,"reg":true,"name":"name",
        "secret":"07d0"}
    ]
}
```

where:

`total`	Total number of pages.
`page`	Current page number.
`records`	Total number of records.

`GET` supports sorting with a default jqgrid syntax. You can use it with paging.

```
GET /admin-api/security/credentials?sidx={x}&sord={y}
```

where:

`sidx` Field to be sorted with values:

- `originalKey`
- `name`
- `key`
- `secret`

`sord` Sorting order with values:

- `asc` (ascending)
- `desc` (descending)

To create a new credential, use the following method. Fields are submitted as form parameters: `key`, `secret`, `auth`, `nw`, `reg`, and `name`.

```
POST /admin-api/security/credentials
```

To update or remove a credential with the provided `key-id`, use the following method. When updating data, the fields are submitted as form parameters: `key`, `secret`, `auth`, `nw`, `reg`, and `name`. The `key-id` in path must match the original key value:

```
PUT|DELETE /admin-api/security/credentials/{key-id}
```

4.8.3 Security eDTLS certificates

To retrieve human-readable, public certificate information of the server, use the following method:

```
GET /admin-api/security/certificate/nsp
```

To retrieve human-readable, public certificate information of the trusted CA, use the following method:

```
GET /admin-api/security/certificate/root
```

To remove all certificate information, use the following method:

```
DELETE /admin-api/security/certificate
```

To set certificate information, use the following method. Data format is plain text of base64 output of `.cer` and `.key` data created and exported using the NanoService Key Tool. The input data must contain a trusted CA public certificate (default alias ca in the NanoService Key Tool), a signed server certificate (default alias `nsp-signed`) and a server private key (default alias `nsp`).

```
POST /admin-api/security/certificate
```

Example:

```
-----BEGIN PRIVATE KEY-----
ME0CAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQcEMzAxAgEBBCDS3kGK3uQ1vh3keRDV
8xecNcyZp9D5sXVeP4jP9GFMb6AKBggqhkjOPQMBBw==
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIBSDCB76ADAgECAgQFe1/XMAoGCCqGSM49BAMCMDQxCzAJBgNVBAYTAkZJMRQwEgYDVQQKEwtB
Uk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMCAXDTE0MDExMDA4MzAxNFoYDzIxMTMxMjE3MDgz
MDE0WjAjMQswCQYDVQQGEwJGSTEUMBIGA1UEChMLQVJNIEZpbmxhbmQwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAAQuBjNUDYkEtpjarHCvERQz2aT8W3eq5SaXylhBugd150UPxXWUphf0CUz+E4pi
9MY9UUpzGeM+v3Q5VnNcD04lMAoGCCqGSM49BAMCA0gAMEUCIQDsb59glhKlYFcuu3AQCoCQcJ0l
zOkyUiCbI08d5iOA5wIgXjlYyjF+GbPTnNs6eQOXoV3F0ZqGut5hRY3ZFml/d7M=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBgDCCASWgAwIBAgIEIZcGAjAMBggqhkjOPQQDAgUAMDQxCzAJBgNVBAYTAkZJMRQwEgYDVQQK
EwtBUk0gRmlubGFuZDEPMA0GA1UEAxMGUm9vdENBMCAXDTE0MDExMDA4MzAxMVoYDzIxMTMxMjE3
MDgzMDExWjA0MQswCQYDVQQGEwJGSTEUMBIGA1UEChMLQVJNIEZpbmxhbmQxDzANBgNVBAMTBlJv
b3RDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABHCtdJ5qFd35pnzW42LPBgHCBrwh9z4LSL+g
kYxfWLhcAKBGMzoNSkf3WSWV9YzRvFBshqZSIPMlV+VlSaAJ8ZWjITAfMB0GA1UdDgQWBBT+Zvp3
VVRwjzIi4ni7u+FG9b3ypTAMBggqhkjOPQQDAgUAA0cAMEQCIBBWEPmziyy2SUWyIrziaaZouln7
EJiJ20MMD0YPCm28AiBAvGSVx7gBliQe1acsGbkKbqLSrlhadFk+5kAKcyaX6Q==
-----END CERTIFICATE-----
```

4.8.4 Client credentials

To retrieve a list of clients, use the following method:

```
GET /admin-api/clients
```

To create a client credential, use the following method:

```
POST /admin-api/clients
```

To update a client's credential (password), use the following method:

```
PUT /admin-api/clients/{domain}/{client-name}
```

4.8.5 Black list

To retrieve blacklisted endpoints, use the following method. No endpoint, with a name that is listed, will be allowed to register. Such an attempt will result in `CoAP 4.03 Forbidden`. In case of certificate-based secure communication, a handshake will be rejected.

```
GET /admin-api/black-list
```

Example:

```
GET /admin-api/black-list

=>
200 OK
Content-Type: text/application+json

["node-1241", "node-4543"]
```

To add endpoints that exist in the system to a black list, use the following method:

```
POST /admin-api/black-list
```

Example:

```
POST /admin-api/black-list
Content-Type: text/application+json

["node-1241", "node-4543"]
=>
204 No Content
```

To remove endpoints from the black list, use the following method:

```
POST /admin-api/black-list?remove=true
```

Example:

```
POST /admin-api/black-list?remove=true
Content-Type: text/application+json

["node-4543"]
=>
204 No Content
```

To check if an endpoint is in the black list, use the following method:

```
GET /admin-api/black-list/{endpoint-name}
```

where:

`Return value`	• `200` Exists. • `404` Does not exist.

To remove an endpoint from the black list, use the following method:

```
DELETE /admin-api/black-list/{endpoint-name}
```

Chapter 5
NanoService C Device Library

This chapter describes how to use the NanoService C Device library and its included examples.

It comprises the following sections:

——— **Note** ———

For detailed function and data structure reference, see the Doxygen documentation included in `installation_directory/doc/Doxygen`.

5.1 Introduction

NSDL C provides the means for a constrained embedded device to become a NanoService endpoint, supporting the IETF *Constrained Application Protocol* (CoAP) over UDP and all interactions with NanoService Platform, or peer-to-peer with other CoAP endpoints. NSDL C is designed to be completely independent from the operating system or Socket API it is used with, and thus can be integrated with most embedded devices.

5.1.1 Overview of features

The minimal configuration of NSDL is provided by the libCoap library, which implements CoAP parsing, building, protocol mechanisms including block transfer and retransmissions, and basic NSP registration. This library can be used on its own to create simple or highly optimized devices applications that minimize the use of Flash and RAM.

A completely automated resource server is provided by the libNSDL library, in addition to libCoAP. This library provides automatic resource handling, NSP registration and observation handling. While taking more Flash and RAM, this library speeds up development.

Finally, NanoService Enterprise includes *enhanced DTLS* (eDTLS) support for end-to-end security with NanoService Platform.

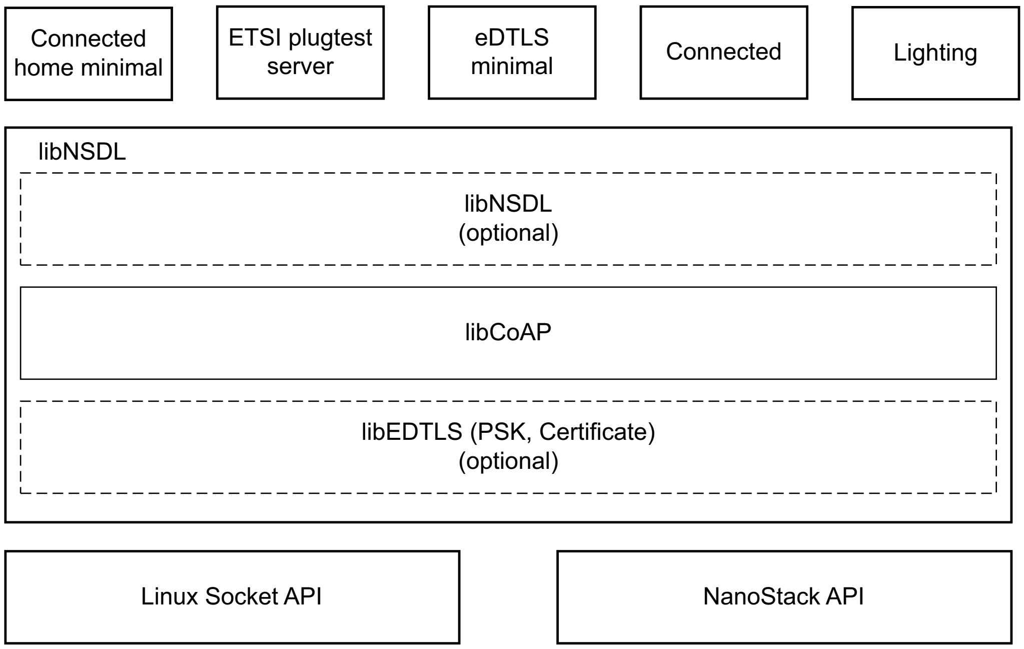


Figure 5-1 NSDL C organization

libCoAP

libCoAP has the following features:

- Comprehensive support for all IETF CoRE standards.
- RFC6690, coap-18, block-14, observe-08.
- Buffer processing model, compatible with any UDP API.
- CoAP Client and Server mode support.
- Basic NSP (resource directory) registration.
- Automated block transfer and retransmission handling.

libNSDL

In addition to the features of libCoap, the addition of libNSDL provides a fully automated embedded Web resource server with the following features:

- Fully automated resource server (static and dynamic resources).
- Easier and faster application development.
- Automated NSP (resource directory) registration.
- RFC6690 resource description engine.
- Automated observation handling.

libEDTLS

NanoService Security is provided by Datagram Transport Layer Security. Sensinode provides an enhanced version of DTLS (eDTLS) as part of the NanoService Device Library. libEDTLS is included with NanoService Enterprise versions of NSDL C:

- eDTLS protocol support in client mode (can initiate a handshake).
- Pre-Shared Key mode support with `TLS_PSK_WITH_AES_128_CCM_8`:
  - — Run-time key configuration.
- Certificate mode with `TLS_ECDHE_ECDSA_WITH_AES128_CCM_8`:
  - — Run-time certificate chain configuration.
- Software AES library when no AES hardware support is available.
- Support for multiple sessions.
- Buffer processing model, compatible with any UDP API.

5.1.2 Resource Requirements

Sizes in are from the MSP430 IAR platform port (application and dynamic RAM not included).

Table 5-1 Resource requirements

Component	Library	Flash	Static RAM
Minimal NSDL	Minimal libCoAP	3.2KB	17B
Basic NSDL	libCoAP	4.9KB	50B
Advanced NSDL	libCoAP and libNSDL	11.1KB	117B
eDTLS	libEDTLS Plus Crypto libraries required for the DTLS mode and cipher suite.	2.4KB	121B

5.1.3 Supported ports

The following binary ports are included in this version of NSDL C. As NanoService Enterprise includes source code for NSDL, porting to other architectures and tool-chains is possible:

- x86 Linux GCC.
- TI MSP430 IAR for NanoStack.
- Atmel RFR2 IAR for NanoStack.

5.2 libCoap

NSDL C consists of a full-featured CoAP library that can easily be integrated with any kind of UDP socket interface and a set of example servers. CoAP support in NSDL C is provided by the libCoap library, which consists of three functional parts:

1. CoAP protocol
   - Handles confirmable CoAP messages resending and acknowledgement.
   - Checks validity of received CoAP message header part.
   - Provides Resource Observation support.
   - Provides Block Transfer support.
2. CoAP message building
   - Builds Packet data from User's given CoAP message structure.
3. CoAP message parsing
   - Parses CoAP message structure from received Packet data.

There are following approaches to using libCoap:

- The complete protocol library is used: CoAP protocol, CoAP message builder and CoAP message parser are included in libCoap.
- Just header parsing and building parts are used: CoAP message builder and CoAP message parser are included in libCoap.

For reducing code size, the following features can be left from compiling with compilation switches:

- Message resending.

  (Define name: `ENABLE_RESENDINGS`)
- Message duplication detection.

  (Define name: `SN_COAP_DUPLICATION_MAX_MSGS_COUNT`)
- Message block transfer.

  (Define name: `SN_COAP_BLOCKWISE_MAX_PAYLOAD_SIZE`)

For inclusion in a project, there are two Header files that define the needed functions:

- `sn_coap_protocol.h`
- `sn_coap_header.h`

Table 5-2 lists the libCoap sn_coap_protocol.h functions and CoAP protocol features:

Table 5-2 CoAP protocol features

Function	Description
`sn_coap_protocol_init()`	This function sets the memory allocation and deallocation functions the library will use, and must be called first.
`sn_coap_protocol_destroy()`	Frees all allocated memory in libCoap protocol part.
`sn_coap_protocol_build()`	Use to build an outgoing message buffer from a CoAP header structure.
`sn_coap_protocol_parse()`	Use to parse an incoming message buffer to a CoAP header structure.

Table 5-2 CoAP protocol features (continued)

Function	Description
`sn_coap_protocol_exec()`	Called periodically to allow the protocol to update retransmission timers and destroy unneeded data.
`sn_coap_protocol_set_block_size()`	If block transfer is enabled, this function changes the block size.
`sn_coap_protocol_set_duplicate_buffer_size()`	If duplicate message detection is enabled, this function changes buffer size.
`sn_coap_protocol_set_retransmission_parameters()`	If re-transmissions are enabled, this function changes resending count and interval.
`sn_coap_protocol_set_retransmission_buffer()`	If re-transmissions are enabled, this function changes resending buffer sizes.
`sn_coap_register()`	Create an NSP registration message.
`sn_coap_register_update()`	Create an NSP update message.
`sn_coap_deregister()`	Create an NSP de-registration message.

Table 5-3 lists the libCoap `sn_coap_header.h` functions and CoAP build and parse features:

Table 5-3 CoAP features

Function	Description
`sn_coap_builder_and_parser_init()`	This function sets the memory allocation and deallocation functions the library will use, and must be called first.
`sn_coap_builder()`	Use to build an outgoing message buffer from a CoAP header structure.
`sn_coap_build_response()`	Use to automate the building of a response to an incoming request.
`sn_coap_parser()`	Use to parse an incoming message buffer to a CoAP header structure.
`sn_coap_packet_debug()`	CoAP packet debugging.
`sn_coap_builder_release_allocated_send_msg_mem()`	This function releases any memory allocated in `sn_nsdl_transmit_s`
`sn_coap_parser_release_allocated_coap_msg_mem()`	This function releases any memory allocated by a CoAP message structure.
`sn_coap_builder_calc_needed_packet_data_size()`	Use to calculate the needed message buffer size from a CoAP message structure.

5.2.1 Using libCoap to manually build a message

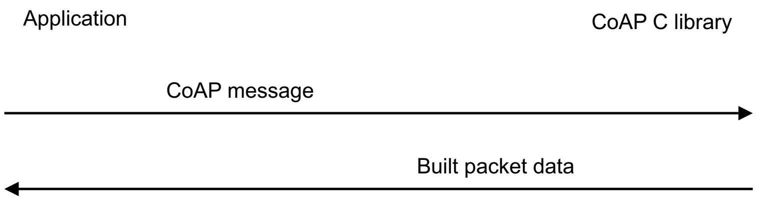


Figure 5-2 Manual CoAP message

User allocates memory for built Packet data and also takes care of releasing that memory.

Required memory count for Packet data is solved by using the `sn_coap_builder_calc_needed_packet_data_size()` function.

Having libCoap build a message

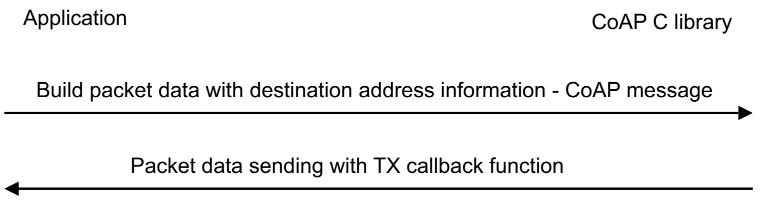

Application
CoAP C library
Build packet data with destination address information - CoAP message
Packet data sending with TX callback function

Figure 5-3 libCoap builds message

User also takes care of releasing memory of Packet data which is built independently by CoAP C-library.

User can release using the `sn_coap_builder_release_allocated_coap_msg_mem()` function.

5.2.2 Using libCoap to parse an incoming message

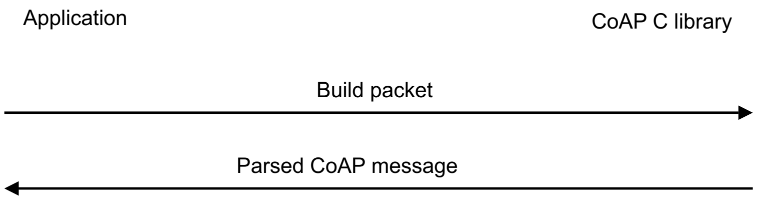


Figure 5-4 libCoap parses message

libCoap allocates memory for CoAP message structure (equal to `sn_coap_hdr_s`) where Packet data is parsed and returned to User.

- User responsibility is to release allocated memory when does not need it any more. User can release with the `sn_coap_parser_release_allocated_coap_msg_mem()` function.
- libCoap does not allocate new memory for Payload part because CoAP message structure's Payload pointer is just pointed to Payload part in Packet data.

  The exception is Blockwise message for which libCoap allocates memory by itself. That happens after all Blockwise messages have arrived and libCoap can gather the whole Payload from the received Blockwise messages.

5.3 libNSDL

NSDL also includes a fully automated resource server (like a small embedded web server), provided by the libNSDL library. This library makes use of libCoap and adds the following additional functionality:

- Automatic link description generation.
- Automated NSP registration.
- Outgoing request support with retransmission handling.
- Automatic handling of incoming resource requests.

To include libNSDL in an application in addition to libCoap, include the header `sn_nsdl_lib.h`, which provides the functions listed in Table 5-4:

Table 5-4 sn_nsdl_lib.h functions

Function	Description
`sn_nsdl_init()`	Initializes libNSDL and must be called first.
`sn_nsdl_create_resource()`	Adds a resource to the server.
`sn_nsdl_delete_resource()`	Removes a resource from the resource server.
`sn_nsdl_exec()`	Must be called periodically for the library to update retransmissions.
`sn_nsdl_get_capability()`	-
`sn_nsdl_get_resource()`	-
`sn_nsdl_get_version()`	-
`sn_nsdl_is_ep_registered()`	Checks if an endpoint is registered.
`sn_nsdl_list_resource()`	Lists the resources of the server.
`sn_nsdl_nsp_lost()`	Sets the registration status to not registered.
`sn_nsdl_process_coap()`	Process an incoming CoAP message.
`sn_nsdl_register_endpoint()`	Registers the endpoint with NSP.
`sn_nsdl_send_coap_message()`	Send an outgoing CoAP request.
`sn_nsdl_send_observation_notification()`	Send a notification message for an ongoing observation.
`sn_nsdl_unregister_endpoint()`	Unregister the endpoint from NSP.
`sn_nsdl_update_registration()`	Update the registration with NSP.
`sn_nsdl_update_resource()`	Update a resource value.
`set_NSP_address()`	Sets NSP address.

5.4 libEDTLS

The eDTLS security library can optionally be used together with either libCoAP alone or with libCoAP and libNSDL. This library provides the following functionality:

- Configure the pre-shared key to use.
- Set and update certificates.
- Connect and disconnect from an eDTLS server.
- Send and receive encrypted data frames.

To include libEDTLS in an application, include the header `sn_edtls_lib.h`, which provides the functions listed in Table 5-5:

Table 5-5 sn_edtls_lib.h functions

Function	Description
`sn_edtls_libraray_initialize ()`	Initializes the eDTLS library. This must be done before connecting to a server.
`sn_edtls_connect ()`	Connects to an eDTLS server.
`sn_edtls_disconnect ()`	Disconnects from an eDTLS server.
`sn_edtls_exec ()`	Must be called evenly during ECC connection. Handles re-sending of the handshake messages and ECC and PRF calculations. Calling this is not required in PSK mode and after establishing the eDTLS session (edtls session status =`EDTLS_CONNECTION_OK`).
`sn_edtls_write_data ()`	Sends encrypted data to the server.
`sn_edtls_parse_data ()`	Decodes encrypted data from the server.
`edtls_pre_shared_key_set ()`	Configures the PSK.
`sn_edtls_destroy ()`	Destroys the eDTLS library. Releases all the allocated memory.
`edtls_certificate_list_update ()`	Updates the library certificate list.
`edtls_set_retransmission ()`	Sets re-transmission parameters for the handshake message.
`sn_edtls_connection_state_query()`	Returns the session state.

5.5 Ports

NSDL is written purely in C, and is portable across different microcontroller platforms, as well as PC platforms. The following ports are currently included with the library:

- x86 Linux.
- TI MSP430.
- Atmel RFR2.

5.5.1 x86 Linux

This port is for x86 based PCs using the standard gcc compiler and is located in the /x86_gcc directory under a library. See the Makefile in the examples projects to see how to include the libraries into a gcc based project.

This port has also been tested on the ARM based Raspberry Pi using its native gcc compiler. This requires the NanoService Enterprise version of NSDL C with source code, as the libraries and examples need to be re-compiled.

5.5.2 TI MSP430

This port is for TI MSP430 micro-controllers and the IAR compiler and is located in the /msp430_iar folder. See the example projects for an example of how to use the libraries on this platform.

This port is not supported.

5.5.3 Atmel RFR2

This port is for the Atmel RFR2 micro-controller and the IAR compiler and is located to `/atmel_rfr2_iar` folder.

5.5.4 Custom ports

If your license include source code of libCoap, it is also possible to port the library to your own microcontroller architecture and compiler toolchain. To do this, follow the following general steps. Specifics will of course depend on your toolchain:

1. Create a new sub-directory `/libCoap/Arch_Compiler` in the project
2. If your toolchain uses make, then copy the `x86_gcc` Makefile and make the needed modifications for you compiler and linker
3. Once the library is compiling, thorough testing and verification will be needed. If port-specific code changes are needed, using suitable `#ifdef #endif` defines is recommended so as not to break other ports.

For some projects, it maybe be more straightforward to include the libCoap headers and code into your project directly instead of building a static library first.

5.6 Examples

Several examples of CoAP servers for use with NanoService Platform are included in the package for x86 Linux with support for IPv6.

5.6.1 Connected Home Example for Linux

This example is a simple command-line Linux CoAP server that registers with NanoService Platform, and de-registers when **Ctrl-C** is pressed. The server emulates a power measurement node with a relay for use in the Connected Home Reference App.

The code consists of a command-line parameter parser in main.c and the CoAP server in connected-home.c. There are two variations of this example:

- The `connected-home_linux` example makes use of the libCoap parsing and building functions along with NSP registration and de-registration functions.
- The `connected-home_full_linux` example makes use of the full libNSDL library for an automated resource server.

To make the example simply:

```
cd nsdl-c-xxx
cd connected-home
make
```

To use the example together with the Connected Home demo setup:

```
./runConnectedHomeDemo.sh
cd nsdl-c-xxx
cd connected-home
./connected-home
```

5.6.2 Lighting Example for Linux

This example is a simple command-line Linux CoAP server that registers with NanoService Platform, and de-registers when **Ctrl-C** is pressed. The server emulates a street lighting node with a relay for use in the Lighting Reference App.

The code consists of a command-line parameter parser in main.c and the CoAP server in lighting.c.

- The `lighting_linux` example makes use of the libCoap parsing and building functions along with NSP registration and de-registration functions.
- The `lighting_full_linux` example makes use of the full libNSDL library for an automated resource server.

To make the example:

```
cd nsdl-c-xxx
cd lighting
make
```

To use the example together with the Connected Home demo setup:

```
./runLightingDemo.sh
cd nsdl-c-xxx
cd lighting
./lighting
```

5.6.3 Connected Home Example for NanoStack

This example is a simple CoAP server running with NanoStack that registers with NanoService Platform. The server emulates a power measurement node with a relay for use in the Connected Home Reference App.

When using `USE_NSP_ADDRESS` flag, example uses hardcoded IPv6 address as a NanoService Platform address. Otherwise application uses routers address to register.

Example includes ports for the MSP430 processor.

5.6.4 Lighting Example for NanoStack

This example is a CoAP server running with NanoStack that registers with NanoService Platform. The server emulates a street lighting node with a relay for use in the Lighting Reference App.

When using `USE_NSP_ADDRESS` flag, example uses hardcoded IPv6 address as a NanoService Platform address. Otherwise application uses routers address to register.

Example includes ports for MSP430 processor:

- The `lighting_nanostack` example makes use of the libCoap parsing and building functions along with NSP registration and de-registration functions.
- The `lighting_nanostack_full` example makes use of the full libNSDL library for an automated resource server.

Chapter 6
NanoService Java Device Library

This chapter describes the requirements and features of the NanoService Java device library.

It comprises the following sections:

6.1 Introduction

This library makes it easy to integrate a Java SE enabled device with the NanoService Platform, or to make an emulation of an embedded device for prototyping and testing purposes.

6.1.1 Overview of features

The following features are supported by the library:

- Complete CoAP support:
  - draft-ietf-core-coap-18.
  - draft-ietf-core-observe-11.
  - draft-ietf-core-block-14.
- CoRE Link Format processing API:
  - RFC6696.
- NanoService Platform registration:
  - As per NSP M2M Interface documentation.
- IPv4 and IPv6 support:
- CoAP server mode (handle incoming requests)
- CoAP client mode (make outgoing requests)
- UDP transport
- TCP transport.

6.1.2 Installing

Note

This version of the NanoService Device Library (NSDL) Java Sun Java JVM v1.6.

You can find the library (`nsdl-java-<version>.jar`) under the release package lib directory.

Place the `nsdl-java-<version>.jar` file under your application and refer the compile/build and runtime classpath to that file. The mechanisms to do that vary depending on your application and development environment.

In a Java application, point the classpath to the `nsdl-java-<version>.jar` when compiling. For example:

- Compile:

```
[user@server ~]# javac -cp lib\nsdl-java-<version>.jar src\*.java
```

- Run:

```
[user@server ~]# java -cp lib\nsdl-java-<version>.jar AppStarter
```

6.2 Example Application

The library includes an example NanoService device application, which creates a CoAP server with resources that can be accessed via the NanoService Platform. Upon initialization, the application automatically registers with the NanoService Platform, and then accepts incoming requests or observations.

The example application can be found from the `/example` directory. This directory contains a ready-made NetBeans project, and can easy be added to NetBeans (**File → Open Project → /example**). The source code is included in the `/example/src` folder within the `com.sensinode.coap.` example package. The needed libraries are included with the example. The built application is available from the `/example/dist` directory. To run the example:

```
cd example/dist
java -jar CoapExample.jar
12:15:20 CoAP server binded on /0.0.0.0:5683
12:15:20 Handler added on /a/relay
12:15:20 Handler added on /s/power
12:15:20 Handler added on /s/temp
12:15:20 Handler added on /s/light
12:15:20 Handler added on /sysUpTime
12:15:20 Handler added on /draft-coap
12:15:20 Handler added on /.well-known/core
```

This will create a CoAP server on the default UDP port 5683 with some example resources. If no address is provided for NSP, then registration is skipped. The default port of NSP is 5683. The default port of the example server, along with the address and port of NSP can be changed with command-line options:

```
java -jar CoapExample.jar -p 8010 -nsp 192.168.0.100:8000
```

The endpoint name when registering is by default `coap-example`, this can be changed using the `-ep` command-line option. For example:

```
java -jar CoapExample.jar -nsp 192.168.0.100 -ep my-server
```

6.2.1 Example code

The example source consists of the main file `CoapExample.java`, which has `main()`, the server start and registration functions.

The other source files implement different kinds of CoAP resources. In the `CoapExample.start()` method a CoapServer object is created and started.

Resources are then attached using the `addRequestHandler()` method.

Finally, registration is performed by creating a Registrator object and using the `register()` method.

Two types of resources are created as request handlers. A few simple static resources are created using the SimpleCoapResource class.

More dynamic resources are created by implementing the CoapResource class, see for example `TestResource.java`. Each of the methods that the resource supports are overloaded, and a suitable response is created and sent.

6.2.2 Building the application

If you make changes to the application, it needs to be re-built so that the `.jar` is updated in the `/dist` directory. This is done by right-clicking on the NetBeans project, and selecting **Clean** and **Build**.

6.3 Creating a Server

This section describes how to create and use a server.

6.3.1 Initializing, starting and stopping the server

To initialize a server, at minimum the port number must be defined. Server parameters must be set before starting a server.

Note

You must use the static method `create()` to make a new instance.

Method `start()` starts server on a new thread.

```
CoapServer server = CoapServer.newBuilder().transport(serverPort).build();

server.start();
```

To stop a server use `stop()` method. It is really important to stop the server

```
server.stop();
```

6.3.2 Adding request handlers

Handlers can be added before or while server is running. There might be several uri paths assigned to the same handler. Handler can be also removed at any time.

```
CoapHandler handler = new SimpleCoapResource("24", "ucum:Cel", 0 );
server.addRequestHandler("/s/temp", handler);
server.addRequestHandler("/temperature", handler);

server.removeRequestHandler(handler);
```

6.3.3 Registering with NSP

Registering the coap server to NanoService Platform is done by the EndPointRegistrator class. It defines all attributes such as endpoint name or domain. It will register and remove registration.

```
EndPointRegistrator registrator = new EndPointRegistrator(server, nspAddress,
"node-001");
registrator.setDomain("test.domain.com");
registrator.register();
…
registrator.unregister();
```

6.3.4 Making CoAP resources

To create CoAP resource, a CoapHandler must be implemented. There is one abstract helper class CoapResource that can be extended. At minimum, the `get()` method must be implemented.

The example below overrides `get()` and `put()` and makes a simple coap resource.

```
public class SimpleCoapResource extends CoapResource {
   private String body="Hello World";

   @Override
   public void get(CoapExchange ex) throws CoapCodeException {
      ex.setResponseBody("Hello World");
      ex.setResponseCode(Code.C205_CONTENT);
      ex.sendResponse();
```

```
    }

    @Override
    public void put(CoapExchange ex) throws CoapCodeException {
             body = ex.getRequestBodyString();
       ex.setResponseCode(Code.C204_CHANGED);
    ex.sendResponse();
    }
}
```

6.4 CoAP Client

For making coap request, use class CoapClient. It uses fluent API. A simple usage example is below:

```
CoapClient client = CoapClient.newBuilder(new InetSocketAddress(
"192.168.0.123",5683)).build();

CoapPacket coapResp = client.resource("/s/temp").sync().get();

coapResp = client.resource("/a/relay").payload("1", MediaTypes.
CT_TEXT_PLAIN).sync().put();

//it is important to close connection in order to release socket
client.closeConnection();
```

Chapter 7
NanoService Java SDK

This chapter describes the requirements and features of the NSP Java SDK for the NanoService Platform.

It comprises the following sections:

7.1 Installation

The *NanoService Platform* (NSP) NSP Java SDK provides a *Java API to NSP* (NSP API) for querying and managing NSP content. The NSP API uses the NSP Web interface for getting and modifying NSP server content.

The NSP Java SDK is a Java library and consists of the jar file `nsp-java-sdk-<version>.jar`.

This version of the NSP Java SDK has the following requirement:

- Sun Java JVM v1.7

The Java library must be added to an application's classpath before it can be used:

1. You can find the NSP Java SDK library (`nsp-sdk-java-<version>.jar`) under the release package lib directory.
2. Place the `nsp-sdk-java-<version>.jar` file under your application and refer the build and runtime classpath to that file. The mechanisms to do that vary depending on your application and development environment.
3. In a Java application, point the classpath to the `nsp-sdk-java-<version>.jar` when running the application. For example:

   ```
   Run:
   [user@server ~]# java -cp lib\nsp-sdk-java-<version>.jar AppStarter
   ```

   In a Java web application, persistent libraries are normally added under the `WEB-INF/lib` folder.

7.2 Introduction to the SDK

This section describes the NSP Java APIs.

7.2.1 NSP connection and queries

The NSP API's basic object is Nsp. Nsp is instantiated by calling the `Nsp.connect()` method. For the connection parameters there are three options:

- Connect to NSP using default settings. See default values in *Configuring the NSP Java SDK* on page 7-8.
- Create your own parameter values by giving the Properties object as a connection parameter.
- Use an external properties file by giving the full path to the file as a connection parameter.

See configuration examples in *Configuring the NSP Java SDK* on page 7-8.

In this example we connect to the NSP Web interface using all default values. After successfully initialization, a new Nsp object is returned.

```
Nsp nspApi = Nsp.connect();
```

For doing an initial request, call the Nsp object's query method.

For a query you may need to provide NspDomain, NspGroup and NspEndpoint objects.

The NspQueryParams object or your implementation of the NspNotificationHandler object can also be provided to the query method for additional functionality.

The NSP Java SDK determines the type of the query from the domain, group, endpoint and additional parameters. Please refer to the NSP Java API Use Cases (tutorials) for more precise query configurations.

The NSP API will always return an NspResult object.

In this example we submit an initial query by giving a domain and endpoint. Name is provided for both domain and endpoint. This example demonstrates querying endpoint resources. The query will return all resources for given endpoint

```
NspResult result = nspApi.query(
   new NspDomain().name("test.domain.com"),
   new NspEndpoint().name("test_ep")
   );
```

This example is the same as above, but the resources are returned by page and result size.

```
NspResult result = nspApi.query(
   new NspDomain().name("test.domain.com"),new NspEndpoint().name("test_ep"),
   new NspQueryParameters().page(2).count(20)
);
```

7.2.2 Working with the result

The NspResult object returned by the NSP API contains the information needed to determine the status and the result data of the query.

The status contains the information if the query was successful or not. The statuses are defined as static integer constants in the Nsp class.

Table 7-1 Successful status values from NspResult

Parameter key	Description
`Nsp.STATUS_OK`	Query was successful and the resulted data is available
`Nsp.STATUS_ACCEPTED`	Nsp has accepted the request from the NSP Java SDK, but was not able to finalize the request and send the response in time
`Nsp.STATUS_NO_CONTENT`	Nsp has made the request from the NSP Java SDK successfully, but the response does not contain any data
`Nsp.STATUS_PARTIAL_CONTENT`	Result does not contain all entries from NSP
`Nsp.STATUS_PARTIAL_SUCCESS`	Group operation (create, update. delete) was only partially successful. In this case the result contains also the failed entries (groups and endpoints) with the result codes.

Table 7-2 Error status

Parameter key	Description
`Nsp.STATUS_NOT_FOUND`	NSP can not find the target (domain, group or endpoint) of the operation and thus the operation can not be performed
`Nsp.STATUS_NOT_AUTHORIZED`	User authentication failed
`Nsp.STATUS_TIMEOUT`	The NSP Java SDK did not get the response from the NSP Web interface in time. Timeout is given as a connection parameter for the NSP Java SDK (see the connection parameters from *Configuring the NSP Java SDK* on page 7-8.
`Nsp.STATUS_SERVER_ERROR`	Undefined Server error from the NSP Web interface

Table 7-3 Partial success result codes for group operations

Parameter key	Description
`Nsp.NspResultCode.DOES_NOT_EXIST`	The group's content entry (group or endpoint) was not found
`Nsp.NspResultCode.SERVER_ERROR`	There was an unknown server error in handling the group's content entry (group or endpoint).

The NSP API returns the data from NSP as a list of NspQuery objects (zero, one or many). NspResult provides an iterator over the collection of NspQuery objects. By using the iterator, you can easily (and in a standard way) go through the list of NspQuery objects and decide what to do with each result.

The type of the resulted query can be checked using the type method of NspQuery. Query types are defined as an NspQueryType enumeration in the Nsp object.

Table 7-4 Query types

Parameter key	Description
`Nsp.NspQueryType.DOMAIN`	the resulted query is embodying a domain
`Nsp.NspQueryType.GROUP`	the resulted query is embodying a group
`Nsp.NspQueryType.ENDPOINT`	the resulted query is embodying an endpoint
`Nsp.NspQueryType.RESOURCE`	the resulted query is embodying an endpoint's resource
`Nsp.NspQueryType.RESOURCEVALUE`	the resulted query is embodying an endpoint resource's value
`Nsp.NspQueryType.UNKNOWN`	the resulted query type can't be determined

This example checks if the status of the NspResult is OK and if so, checks the type of the result, go through the endpoint resources and prints the path to the console.

```
int status = nspResult.status();
   if (status == Nsp.STATUS_OK) {
      Iterator<NspQuery> iterator = nspResult.iterator();
      while (iterator.hasNext()) {
         NspQuery query = iterator.next();
         if (query .type() == Nsp.NspQueryType.RESOURCE) {
            System.out.println(query.getEndpoint().getResource().getPath());
         }
      }
}
```

This example handles the result of a group operation (create, update, or delete). It demonstrates a partially successful result. NspResult contains the failure information for the case. The example prints the failed entries and result codes for the failures.

```
int status = nspResult.status();
if (status == Nsp.STATUS_PARTIAL_SUCCESS) {
   Failures failures = nspResult.getFailures();
   if (failures != null) {
      // end-points
      Iterator<NspEndpoint> endpointIterator = failures.getEndpoints().iterator();
      while (endpointIterator.hasNext()) {
         NspEndpoint endpoint = endpointIterator.next();
         System.out.println("Group operation failed for endpoint ” +
            endpoint.getName() + " with result code: " +
            endpoint.getResultCode());
      }
      // groups
      Iterator<NspGroup> groupIterator =failures.getGroups().iterator();
      while (groupIterator.hasNext()) {
         NspGroup group = groupIterator.next();
         System.out.println("Group operation failed for group " +group.getName() +
               " with result code: " +group.getResultCode());
      }
   }
}
```

It is also possible to 'drill down' the levels of Nsp entries by calling the returned NspQuery object's query method. For example: first query for endpoints of a domain and immediately query also resources for each endpoint.

```
// first query for the domain's endpoints
NspResult nspResult = nspApi.query(new NspDomain().name("test.domain.com"));
// handle the result
if (nspResult.status() == Nsp.STATUS_OK) {
   // get the iterator over endpoints
   Iterator<NspQuery> endpointIterator = nspResult.iterator();
   while (endpointIterator.hasNext()) {
      NspQuery endpointQuery = endpointIterator.next();
      if (endpointQuery .type() == Nsp.NspQueryType.ENDPOINT) {
         // immediately query for endpoint resources
         nspResult = endpointQuery.query();
         if (nspResult.status() == Nsp.STATUS_OK) {
            // get the iterator over resources
            Iterator<NspQuery> resourceIterator = nspResult.iterator();
            while (resourceIterator.hasNext()) {
               NspQuery resourceQuery = resourceIterator.next();
               if (resourceQuery.type() == Nsp.NspQueryType.RESOURCE) {
                 System.out.println(resourceQuery.getEndpoint()
                                      .getResource()
                                      .getPath());
               }
            }
         }
      }
   }
}
```

7.2.3 Exception handling

Exception handling in the NSP Java SDK is separated in two situations:

- Exception handling in initialization of the NSP API
- Exceptions in executing queries with an (already connected) NSP API

At initialization time, the NSP API will throw an NspInitializationException if the initialization of the NSP API fails. You must catch this exception or have it thrown to user classes of your method.

```
// use try-catch to handle the possible initialization error...
try {
   nspApi = Nsp.connect();
} catch (NspInitializationException e) {
      // log exception etc...
}
// .. or throw the exception to the user of your method

public void myMethodToInitNspApi() throws NspInitializationException {
   Nsp nspApi = Nsp.connect();
}
```

When doing queries, use NspResult to check if errors happened. See *Working with the result* on page 7-3. It is not required to catch any exceptions during query execution.

7.3 Configuring the NSP Java SDK

NSP connection parameters

- Control if the connection to the NSP Web interface is tested during NSP API initialization
- Change the request timeout value for executing queries to NSP
- Change the observation timeout for waiting notifications from NSP
- Change the time waited before re-try the initiation of observation
- Change the method of receiving event from NSP. In 'Long polling' events are actively fetched from NSP. Using notification servlet (set up in web container) that handles events sended by NSP.

Table 7-5 NSP parameters and defaults

Parameter key	Description	Default value
`NSP_DOMAIN`	the name of the NSP Web interface domain	
`NSP_PORT`	the port number of NSP Web interface	
`NSP_PROTOCOL`	the protocol used by NSP Web interface, options: 'http', 'https'	HTTP
`NSP_AUTH_DOMAIN`	the domain for authentication to the NSP Web interface	
`NSP_HTTPS_TRUST_KEYSTORE`	Java keystore file containing NSP public certificate. (Mandatory if `NSP_PROTOCOL` is https and NSP uses self signed certificate.)	
`NSP_HTTPS_TRUST_KEYSTORE_PASSWORD`	Java keystore file password. (Mandatory if `NSP_PROTOCOL` is https and NSP uses self signed certificate.) `TEST_CONNECTION` Boolean to determine if the connection to the NSP Web interface is tested during the initialization of the NSP API. If this parameter is set to true and connection to the NSP Web interface fails, an NspInitializationException is thrown.	TRUE
`TEST_CONNECTION`	Boolean to determine if the connection to the NSP Web interface is tested during the initialization of the NSP API. If this parameter is set to true and connection to the NSP Web interface fails, an NspInitializationException is thrown.	TRUE
`REQUEST_TIMEOUT`	timeout for requests to the NSP Web interface in milliseconds. The NSP Java SDK will return a timeout error after this time if no response is submitted from the NSP Web interface.	60 000 (60 seconds)
`OBSERVATION_TIMEOUT`	timeout for observation requests to the NSP Web interface in milliseconds. The NSP Java SDK will reconnect for observation after this time.	120 000 (2 minutes)
`OBSERVATION_SLEEPTIME_IN_EXCEPTION`	Sleeptime before retrying observation after exception occurs in observation request.	20 000 (20 seconds)

Table 7-5 NSP parameters and defaults **(continued)**

Parameter key	Description	Default value
`ASYNC_REQUEST_TIMEOUT`	Timeout value for asynchronous resource request if response does not arrive (in seconds)	600 (10 minutes)
`NOTIFICATION_SERVLET_IN_USE`	'true' defines the use of notification servlet that receives events from NSP. 'false' defines use of long polling.	TRUE
`NOTIFICATION_SERVLET_URL`	Notification servlet url where NPS sends events. For example, where you have configured (in web.xml) the sdk notification servlet. `REMOTE_HOST` means NSP gets IP address from existing connection but it can be set to some specific address if needed for example, if some firewall issues arises.	`http://REMOTE_HOST:8083/nspevents`

For error handling, you must be ready to handle the `NspInitializationException` thrown by the `Nsp.Connect()` method. See the section *Exception handling* on page 7-6 for more details.

The example connects to NSP using default values. In this case the `nsp.properties` file MUST be found under the classpath of the application using the NSP Java SDK. You can find an example of an `nsp.properties` file under the release package of the NSP Java SDK conf directory.

```
Nsp nspApi = Nsp.connect();
```

In this example, we connect to the NSP Web interface using a custom Nsp domain and port. Defaults are used for the rest of the parameters. After successfully connecting to NSP, a new Nsp object is returned.

```
Properties props = new Properties();
props.put("NSP_PORT", "8081");
props.put("NSP_DOMAIN", "mydomain.nsp.rest.com");
Nsp nspApi = Nsp.connect(props);
```

The next example connects to the NSP Web interface using an external properties file.

```
Nsp nspApi = Nsp.connect("/tmp/nsp.properties");
```

7.4 Secured REST connections between NSP and NSP API

For security reasons, it is possible to use secure https connections between NSP and Application using REST interface, (in this instruction using NSP API). This is to be considered if they are running in different environments and traffic goes through non secure connection outside private network.

Java keystore file is used for storing certificates and trusted public key. Keystore file is manipulated using java keytool command, for more details see `http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`

The pre-configured example of keystores and configurations should not be used in product environment, they are only for testing environments with secure connection.

7.4.1 Configuring SSL connections

There are two separate http connections between NSP and NSP API that can be independently configured to use secure https connection:

- Server in NSP where NSP API makes connection and sends outgoing messages and receives responses to those messages.
- Servlet in NSP API that receives observations that are sent actively from NSP (notification servlet).

Securing NSP Server connection

1. NSP Configuration:
   a. NSP must be started in https mode (`com.sensinode.server.https.in.use=true`).
   b. NSP keystore file must be set correctly, for example:
   `'com.sensinode.server.https.keystore.file=../conf/nsp.keystore'`
   c. NSP keystore file password must be set correctly in obfuscated format:
   `'com.sensinode.server.https.keystore.password=OBF:1tvd19q71v231ktv1ktz1l161kxw1kqv1kqz1v2b19q71tvp'`
   Clear text password can be obfuscated by running:
   `org.mortbay.jetty.security.Password`
   as a main class.
   See the *Password issues* section of:
   `http://wiki.eclipse.org/Jetty/Howto/Configure_SSL`
   for more information about obfuscation.
2. 1.2. NSP API Configuration in nsp properties:
   a. `NSP_PROTOCOL` must be '`https`'.
   b. `NSP_HTTPS_TRUST_KEYSTORE` must point to valid trusted keystore containing public key of NSP certificate.
   c. `NSP_HTTPS_TRUST_KEYSTORE_PASSWORD` must be set correctly as clear text of trusted keystore password.

Securing NSP API servlet for notification servlet connection

These steps are required if using self signed certificate in configured web container:

1. NSP Configuration:
   a. NSP trusted keystore file must be set correctly, for example:

```
'com.sensinode.server.https.trusted.keystore.file=../conf/trust.keystore'
```

This trusted keystore must contain public key of NSP API certificate (example found in jetty-ssl.keystore in reference applications)

b. NSP trusted keystore password must be set correctly as clear text:

```
'com.sensinode.server.https.trusted.keystore.password=d4hjln346l4j'
```

2. NSP API Configuration

   a. Implemented web container is started in SSL mode (examples in reference web application startup scripts and web.xml)

7.4.2 Creation of new certificate keystore files

This sectin explains certificate creation using the keytool executable shipped in with Java JRE/SDK. Procedure is essentially identical for both NSP and NSP API notification servlet, this example is for NSP connection. To make NSP API subscription server certificate, only the files are put other way around comparing NSP connection setup and the created keystore is used as web containers keystore:

1. Create new certificate to a new keystore:

   ```
   > keytool -genkeypair -alias myalias -keypass mypassword -keystore
   mykeystore.keystore
   ```

   This produces keystore file described in *Configuring SSL connections* on page 7-10.

2. Get the obfuscated password from keystore password with `org.mortbay.jetty.security.Password`, this produces obfuscated keystore password described in *Configuring SSL connections* on page 7-10.

3. Export certificate from keystore:

   ```
   > keytool -export -alias myalias -file mycert.crt -keystore mykeystore.keystore
   ```

4. Create new keystore for trusted certificate:

   ```
   > keytool -import -trustcacerts -alias myalias -file mycert.crt -keystore
   mytrust.keystore
   ```

   This produces trusted keystore file. Remember the password given in keystore creation. That is the field for trusted keystore password described in *Configuring SSL connections* on page 7-10.

Chapter 8
NanoService Java Use Cases

This chapter describes various use cases for the NSP Java SDK.

It comprises the following sections:

8.1 Introduction

The NSP Java SDK provides an easy way for web app developers to make use of the NSP Web interfaces (see the NSP Web Interface documentation for more information). The Java SDK integrates directly into any JSP, Java SE or Java EE project, and provides easy to use Java API for interacting with NSP.

The rest of this document covers different NSP use cases. See the Java Docs for detailed specifications of the API. Also see the NanoService Reference Application code which serves as a reference for using the SDK in a complete solution.

This version of the NanoService Platform (NSP) Java SDK has the following requirement:

- Sun Java JVM v1.6.

8.2 Initializing the API

First initialize NSP API. Note that in these use case examples we use the default connection parameters for NSP. This needs to be done once when starting to use the API for a NSP domain.

```
Nsp nspApi = Nsp.connect(); // default connection parameters are used
```

8.3 Entry queries

This section describes how to use queries.

8.3.1 Query for domains

Prerequisites: none

The NSP Web interface will receive the following XRI lookup request:

```
GET /xri/?_xrd_r=text/url-list;sep=true
```

Execute the query by calling the query -method in the NSP API. There is no need to specify additional parameters, since we are querying the 'top level' that is, the domains.

```
NspResult nspResult = nspApi.query();
```

If you want to limit the result size, provide query parameters page and count.

```
 NspResult nspResult = nspApi.query(
    new NspQueryParameters().page(1)  // NOTE: page not mandatory.
    .count(20)  // If page not given,
                // NSP will return first 20 elements
);
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the names of domains received.

```
 if (nspResult.status() == Nsp.STATUS_OK) {
     Iterator<NspQuery> iterator = nspResult.iterator();
    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.DOMAIN) {
            System.out.println("Domain found with name: " +
                 query.getDomain().getName());
        }
    }
}
```

8.3.2 Query for endpoints of a domain

Prerequisites: To look up a domain's endpoints, you must know the domain name.

The NSP Web interface will receive following XRI lookup request:

```
GET /xri/@test.domain.com?_xrd_r=text/url-list;sep=true
```

Do the query by calling the query method in the NSP API. Provide a name attribute for the domain.

```
 NspResult nspResult = nspApi.query(new NspDomain().name("test.domain.com"));
```

If you want to limit the result size, provide query parameters page and count.

```
 NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspQueryParameters().page(1) // NOTE: page not mandatory.
                            .count(20) // If page not given, Nsp will return
                                       // first 20 elements);
```

As a result we get the NspResult object. In this example we check the status, iterate through the resulted queries and show the names of endpoints received.

```
 if (nspResult.status() == Nsp.STATUS_OK) {
    Iterator<NspQuery> iterator = nspResult.iterator();
    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.ENDPOINT) {
            System.out.println("End-point found with name: " +
                    query.getEndpoint().getName());
        }
    }
}
```

8.3.3 Query for endpoint resources

Prerequisites: To look up an endpoint's resources, you must know the name of the domain where the endpoint is located and the endpoint's name.

The NSP Web interface will receive the following XRI lookup request:

```
GET /xri/@test.domain.com*entry1?_xrd_r=text/url-list;sep=true
```

Do the query by calling the query -method in the NSP API. Provide the name attribute for the Domain and endpoint.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspEndpoint().name("entry1")
);
```

If you want to limit the result size, provide query parameters page and count.

```
NspResult nspResult = nspApi.query(
            new NspDomain().name("test.domain.com"),
            new NspEndpoint().name("entry1"),
            new NspQueryParameters()
                .page(1)     // NOTE: page not mandatory.
                .count(20)   // If page not given, Nsp will return
                             // first 20 elements
);
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the path of endpoint resources received.

```
if (nspResult .status() == Nsp.STATUS_OK) {
         Iterator<NspQuery> iterator = nspResult.iterator();

    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.RESOURCE) {
            System.out.println("Resource found with name: " +
            query.getEndpoint().getResource().getPath());
        }
    }
}
```

8.3.4 Search for entries

Prerequisites: To search for either or both endpoints and resources, you must know the name of the domain where the endpoint is located and some data for either or both the endpoint and its resource.

The NSP Web interface will receive the following XRI lookup request:

```
GET /xri/@test.domain.com?ep-type=[endpointtype] &
    type=[resourcetype]&_xrd_r=text/url-list;sep=true
```

Execute the searches by calling the query -method in the NSP API. Give a name for the domain, search criteria for endpoint and resource.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspEndpoint().type("Temperature").resourceType("resourcetype")
);
```

If you want to limit the result size, provide query parameters `page` and `count`.

```
NspResult nspResult = nspApi.query(
    new NspQueryParameters().page(1)    // NOTE: page not mandatory.
                            .count(20)  // If page not given,
                                        // NSP will return first 20 elements
);
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the name of the resources found.

```
if (nspResult .status() == Nsp.STATUS_OK) {
        Iterator<NspQuery> iterator = nspResult.iterator();
    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.RESOURCE) {
            System.out.println("Resource found with path: " +
                     query.getEndpoint().getResource().getPath());}}}
```

8.3.5 Get metadata for entries

Prerequisites: To query for a domain, endpoint or resource metadata, you must know the name of the domain, endpoint or resource. You can also query a list of entries with metadata, for example, query for all domains with their metadata.

The NSP Web interface will receive an XRI lookup with a query parameter `'application/link-format'`. For example:

```
GET /xri/@test.domain.com*epname?_xrd_r=application/link-format
```

Execute the query by calling the query -method in the NSP API. Give the method NspMethod.GET_METADATA at the level where the metadata is required.

```
 // get all domains with metadata
NspResult nspResult = nspApi.query(
            new NspDomain().method(Nsp.NspMethod.GET_METADATA)
        );

 // get domain's endpoints with metadata
NspResult nspResult = nspApi.query(
            new NspDomain().name("test.domain.com"),
            new NspEndpoint().method(Nsp.NspMethod.GET_METADATA)
        );

// get end-point's resources with metadata
NspResult nspResult = nspApi.query(
            new NspDomain().name("test.domain.com"),
            new NspEndpoint().name("epname").method(Nsp.NspMethod.GET_METADATA)
        );
```

As a result we get an NspResult object. In this example we check the status and iterate through resulted queries and show some of the resource's metadata.

```
if (nspResult.status() == Nsp.STATUS_OK) {
     Iterator<NspQuery> iterator = nspResult.iterator();
```

```
        while (iterator.hasNext()) {
            NspQuery query = iterator.next();
            if (query.type() == Nsp.NspQueryType.RESOURCE) {
                NspResource resource =query.getEndpoint().getResource();
                System.out.println("Sesource found with metadata: " +", path=" +
                    resource.getPath() +", description=" + resource.getDescription() +
                    ", type=" + resource.getType()
                );
            }
        }
    }
```

8.4 Grouping

This section describes how to use groups.

8.4.1 Get groups for a domain

Prerequisites: To query for a domain's groups, you must know the name of the domain.

The NSP Web interface will receive an XRI lookup with a query parameter 'application/json'.

For example:

```
GET /xri/@test.domain.com/+group?_xrd_r=application/json
```

Execute the query by calling the query -method in the NSP API. Give the `name` attribute and `GET_GROUPS` method for the domain.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com").method(NspMethod.GET_GROUPS)
    );
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the names of groups received.

```
if (nspResult.status() == Nsp.STATUS_OK) {
         Iterator<NspQuery> iterator = nspResult.iterator();
        while (iterator.hasNext()) {
            NspQuery query = iterator.next();
            if (query.type() == Nsp.NspQueryType.GROUP) {
                System.out.println("Group found with name: " +
                     query.getGroup().getName());}}}
```

8.4.2 Get group's contents

Prerequisites: To query for a domain's groups, you must know the name of the group.

The NSP Web interface will receive an XRI lookup with a query parameter '`application/json`'.

For example:

```
GET /xri/@test.domain.com/+group:testgroup?_xrd_r=application/json
```

Execute the query by calling the query -method in the NSP API. Give the `name` attribute and `GET_GROUPS` method for the domain.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),new NspGroup().name("testgroup")
    );
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the names of endpoints and groups received.

```
if (nspResult.status() == Nsp.STATUS_OK) {
        Iterator<NspQuery> iterator = nspResult.iterator();
        while (iterator.hasNext()) {
            NspQuery query = iterator.next();
            if (query.type() == Nsp.NspQueryType.GROUP) {
                System.out.println("Group found with name: " +
                     query.getGroup().getName());
                if (query.getGroup().getGroups() != null) {
                    for (NspGroup gr : query.getGroup().getGroups()) {
                        System.out.println("SubGroup found " + gr.getName());
                    }
                }
                if (query.getGroup().getEndpoints() != null) {
                    for (NspEndpoint ep : query.getGroup().getEndpoints()) {
                        System.out.println("EndPoint found " + ep.getName());
                    }
                }
            }
        }
    }
```

8.4.3 Search group's endpoints

Prerequisites: To search for a group's endpoints, you must know the name of the domain where the endpoint is located and some data for the endpoint or its resources.

The NSP Web interface will receive the following kind of XRI lookup requests:

```
GET /xri/@test.domain.com/+group:mygroup/temp (search with resource path)
GET /xri/@test.domain.com/+group:mygroup?if=ns.wadl#b (search with resource
         interface)
```

Execute the search by calling the query -method in the NSP API. Give a name for the domain, define the group with endpoint and add search criteria for group's endpoint or resource.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspGroup().name("mygroup").withEndpoint(
            new NspEndpoint().type("ep-tyyppi").resourceType("application/json")
        )
);
```

As a result we get an NspResult object. In this example we check the status and iterate through the resulted queries and show the name of the endpoints found.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    Iterator<NspQuery> iterator = result.iterator();
    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.ENDPOINT) {
            System.out.println("Endpoint found with name: " +
            query.getEndpoint().getEndpoint().getName());
        }
    }
}
```

8.4.4 Create new group

Prerequisites: To create a new group, you must know the domain the group is created into. If you wish to add content for the group, you must know the names for either or both the endpoints and subgroups which are added to the new group. The endpoints and subgroups MUST exist in NSP before they can be added under the new group.

The NSP Web interface will receive the request:

```
POST /xri/@test.domain.com/+group:mynewgroup/
```

The group's content is defined in the The NSP Web interface request body.

Execute the query by calling the `query()` method.

If a group with no content is created, give name for the domain and the group. Give the CREATE method for the group.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspGroup().name("mynewgroup").method(NspMethod.CREATE)
    );
```

If a group with content is created, give name for the domain and the group. Give the `CREATE` method for the group. Define the endpoints and subgroups for the group using `withGroup` and `withEndpoint` methods.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspGroup().name("myNewGroup")
                    .description("main group")
                    .method(NspMethod.CREATE).withGroup(
                        new NspGroup().name("myNewSubGroup1")
                        )
                    .withGroup(
                        new NspGroup().name("myNewSubGroup2")
                        )
                    .withEndpoint(
                        new NspEndpoint().name("newEp4Group1")
                        )
                    .withEndpoint(
                        new NspEndpoint().name("newEp4Group2")
                        )
);
```

As a result we get an NspResult object. if the group was created, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {System.out.println("Group created");}
```

If adding group's content was not (at all or partially) successful, `STATUS_PARTIAL_SUCCESS` is returned. In this case, the result contains the failure information, that is either or both the failed endpoints and groups with names and the failure result codes.

```
if (nspResult.status() == Nsp.STATUS_PARTIAL_SUCCESS) {
    Failures failures = nspResult.getFailures();
    if (failures != null) {
        // end-points
        Iterator<NspEndpoint> endpointIterator =
            failures.getEndpoints().iterator();
        while (endpointIterator.hasNext()) {
            NspEndpoint endpoint = endpointIterator.next();
            System.out.println("Adding endpoint " + endpoint.getName() +
                " to the group failed with result code: " +endpoint.getResultCode());
        }
        // groups
        Iterator<NspGroup> groupIterator =failures.getGroups().iterator();
        while (groupIterator.hasNext()) {
            NspGroup group = groupIterator.next();
            System.out.println("Adding group " + group.getName() +
                " to the group failed with result code: " +group.getResultCode());
        }
    }
}
```

8.4.5 Update a group

Prerequisites: To update a group, you must know the domain and the group. If you wish to add content for the group, you must know the names for either or both the endpoints and subgroups which are added to the existing group. The endpoints and subgroups MUST exist in NSP before they can be added under the existing group.

The NSP Web interface will receive the request:

```
PUT /xri/@test.domain.com/+group:myoldgroup/
```

The group's content is defined in the The NSP Web interface request body.

Give name for the domain and the group. Give the `UPDATE` method for the group. Define the endpoints and subgroups for the group using `withGroup` and `withEndpoint` methods.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspGroup().name("myoldgroup").description("new description")
                        .method(NspMethod.UPDATE)
                        .withGroup(
                            new NspGroup().name("newSubGroup")
                        )
                        .withEndpoint(
                            new NspEndpoint().name("newEp")
                        )
);
```

As a result we get an NspResult object. if the group was updated, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {System.out.println("Group updated");}
```

If updating group's content was not (at all or partially) successful, `STATUS_PARTIAL_SUCCESS` is returned. In this case the result contains the failure information, that is either or both the failed endpoints and groups with names and the failure result codes.

```
if (nspResult.status() == Nsp.STATUS_PARTIAL_SUCCESS) {
    Failures failures = nspResult.getFailures();
    if (failures != null) {
        // end-points
        Iterator<NspEndpoint> endpointIterator =failures.getEndpoints().iterator();
        while (endpointIterator.hasNext()) {
            NspEndpoint endpoint = endpointIterator.next();
            System.out.println("Updating endpoint " +
                endpoint.getName() +" for the group failed with result code: "
                + endpoint.getResultCode());
        }
        // groups
        Iterator<NspGroup> groupIterator =failures.getGroups().iterator();
        while (groupIterator.hasNext()) {
            NspGroup group = groupIterator.next();
            System.out.println("Updating group " +group.getName() +
                " for the group failed with result code: " +group.getResultCode());
         }
    }
}
```

8.4.6 Delete a group

Prerequisites: To delete a group, you must know the domain and the group you want to delete.

The NSP Web interface will receive the request:

```
DELETE /xri/@test.domain.com/+group:mygroup/
```

Execute the query by calling the `query()` method. Give name for the domain and the group. Give the `DELETE` method for the group.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspGroup().name("mygroup").method(NspMethod.DELETE)
);
```

As a result we get an NspResult object. If the group was deleted successfully, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {System.out.println("Group was deleted");}
```

8.4.7 Delete group's content

Prerequisites: To delete a group's content, you must know the domain, the group and the content (that is, groups and endpoints) you want to delete.

The NSP Web interface will receive the request:

```
DELETE /xri/@test.domain.com/+group:mygroup/
```

The group's content is defined in the The NSP Web interface request body .

Execute the query by calling the query -method. Give name for the domain and the group. Give the `DELETE` method for the group. Define the deleted endpoints and subgroups for the group using withGroup and withEndpoint -methods.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspGroup().name("mygroup").method(NspMethod.DELETE)
        .withEndpoint(new NspEndpoint().name("delete-me-endpoint"))
);
```

As a result we get an NspResult object. If the group was deleted successfully, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Group content was deleted");
}
```

If deleting group's content was not (at all or partially) successful, `STATUS_PARTIAL_SUCCESS` is returned. In this case, the result contains the failure information, that is either or both the failed endpoints and groups with names and the failure result codes.

```
if (nspResult.status() == Nsp.STATUS_PARTIAL_SUCCESS) {
    Failures failures = nspResult.getFailures();
    if (failures != null) {
        // end-points
        Iterator<NspEndpoint> endpointIterator =
            failures.getEndpoints().iterator();
        while (endpointIterator.hasNext()) {
            NspEndpoint endpoint = endpointIterator.next();
            System.out.println("Deleting endpoint " +
                endpoint.getName() +" from the group failed with result code: "
                 +endpoint.getResultCode());
        }
        // groups
        Iterator<NspGroup> groupIterator =failures.getGroups().iterator();
        while (groupIterator.hasNext()) {
            NspGroup group = groupIterator.next();
            System.out.println("Deleting group " +group.getName() +
            " from the group failed with result code: " +group.getResultCode());
        }
    }
}
```

8.4.8 Change resource values of group's content

Note

To change a resource's value, you must know the domain, group and resource path.

The NSP Web interface will receive the request:

```
PUT /proxy/+group:mygroup.test.domain.com/sensor/temperature
```

Execute the query by calling the `query()` method. Give a name for the domain and the group and resource.

```
NspResource resource = new NspResource()
        .path("/sensor/temperature")
        .value("new value")
        .contentType("text/plain");
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspGroup().name("mygroup").resource(resource).method(NspMethod.UPDATE));
```

As a result we get an NspResult object. If `STATUS_OK` is returned then the result can be iterated to get results of separate endpoints.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    int resultValue = 0;
    Iterator<NspQuery> iterator = result.iterator();
    while(iterator.hasNext()) {
        if (iterator.next().getStatus() != 200) {
            resultValue++;
        }
    }
}
```

8.5 Proxy queries

This section describes proxies.

8.5.1 Get an endpoint resource value

Prerequisites: To retrieve an endpoint's resource value, you must know the name of the domain where the endpoint is located, the endpoint's name and its resource path.

The NSP Web interface will receive the following request:

```
GET /proxy/entry1.test.domain.com/sensor/temperature
```

Execute the query by calling the query -method in the NSP API. Provide the `name` attribute for the Domain and endpoint. For the resource you need to give the path.

```
 NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint().name("entry1").resourcePath("sensor/temperature")
);
```

As a result we get an NspResult object. In this example we check the status and iterate through the queries and show the values of resources received.

```
if (nspResult.status() == Nsp.STATUS_OK) {
     Iterator<NspQuery> iterator = nspResult.iterator();
    while (iterator.hasNext()) {
        NspQuery query = iterator.next();
        if (query.type() == Nsp.NspQueryType.RESOURCEVALUE) {
            System.out.println("Resource value received: "
                 +query.getEndpoint().getResource().getValue());
        }
    }
}
```

8.5.2 Change a resource value

Prerequisites: To change a resource's value, you must know the domain, endpoint and resource path.

The NSP Web interface will receive the request:

```
PUT /proxy/test_ep.test.domain.com/sensor/temperature
```

Execute the query by calling the `query()` method. Give a name for the domain and the endpoint and path for the resource. Set the resource's new value.

```
NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspEndpoint()
            .name("test_ep").resourcePath("/sensor/temperature")
            .resourceValue("new value")
);
```

As a result we get an NspResult object. if the value was changed, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
        System.out.println("Resource value changed");
}
```

If resource value is binary data set value as byte array and use content type application/octet-stream

```
byte[] payload = "some new binary value".getBytes();
NspResult nspResult = nspApi.query(
      new NspDomain().name("test.domain.com"),
      new NspEndpoint().name("test_ep")
        .resourcePath("/foo/bar")
        .resourceByteValue(payload)
        .resourceContentType("application/octet-stream")
);
```

8.5.3 Change a resource values of group's content

See *Grouping* on page 8-8 for how to change a resource values of group's content.

8.5.4 Create a new resource

Prerequisites: To create a new resource, you must know the domain and endpoint.

The NSP Web interface will receive the request:

```
POST /proxy/test_ep.test.domain.com/sensor/newresource
```

Execute the query by calling the `query()` method. Give name for the domain and the endpoint.

Give the `CREATE` method for the endpoint.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint()
        .name("test_ep")
        .resourcePath("/sensor/newresource")
        .method(NspMethod.CREATE)
);
```

As a result we get an NspResult object. if the resource was created, `STATUS_OK` is returned.

```
 if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Resource created");
}
```

8.5.5 Delete a resource

Prerequisites: To delete a resource, you must know the domain, endpoint and the resource you want to delete.

The NSP Web interface will receive the request:

```
DELETE /proxy/test_ep.test.domain.com/sensor/temperature
```

Execute the query by calling the `query()` method. Give name for the domain and the endpoint.

Give the `DELETE` method for the endpoint.

```
 NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspEndpoint()
              .name("test_ep")
              .resourcePath("/sensor/temperature")
              .method(NspMethod.DELETE)
);
```

As a result we get an NspResult object. If the resource was deleted successfully, `STATUS_OK` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
        System.out.println("Resource was deleted");
}
```

8.6 Asynchronous proxy query

Asynchronous query uses notification channel for receiving response from NSP. It supports `GET`, `PUT`, `DELETE`, `POST` methods.

8.6.1 Get an endpoint resource value

Prerequisites: To retrieve an endpoint's resource value, you must know the name of the domain where the endpoint is located, the endpoint's name and its resource path.

The NSP Web interface will receive the following request:

```
GET /async-proxy/entry1.test.domain.com/sensor/temperature
```

Implement the AsyncListener interface and execute the query by calling the `asyncQuery()` method in the NSP API.

```
AsyncListener myListener = new MyAsyncListenerImpl();
NspResult nspResult = nspApi.asyncQuery("test.domain.com", "entry1",
    "/sensor/temperature", null, "GET", null, myListener);
```

Listener will get call on `complete()` method, where result is received:

```
if (code == Nsp.STATUS_OK && payload != null) {
    String payloadString = new String(payload);
}
```

8.6.2 Change a resource value

Prerequisites: To change a resource's value, you must know the domain, endpoint and resource path.

The NSP Web interface will receive the request:

```
PUT /async-proxy/test_ep.test.domain.com/sensor/temperature
```

Execute the query by calling the `query()` method. Give a name for the domain and the endpoint and path for the resource. Set the resource's new value.

```
AsyncListener myListener = new MyAsyncListenerImpl();
NspResult nspResult = nspApi.asyncQuery("test.domain.com", "test_ep",
    "/sensor/temperature", null, "PUT", "new value".getBytes(), myListener);
```

Listener will get call on `complete()` method.

```
complete(int code, byte[] payload, Long maxAge, String contentType,
         String errorMessage) {
            if (code == Nsp.STATUS_OK) {
                System.out.println("Resource value changed");
            }
}
```

8.7 Event handling

This section describes how to handle resource events.

8.7.1 Subscribe to a resource

Prerequisites: To subscribe to a resource, you must know the domain, endpoint and resource to which you want to subscribe.

The NSP Web interface will receive an event request:

```
PUT /events/subscription/resource/test_ep.test.domain.com/sensor/temperature
```

Add the subscription to a resource by calling the `query()` method in the NSP API. Give a name for the domain and the endpoint and path for the resource. Set the endpoint's method as `SUBSCRIBE`.

```
 NspResult nspResult = nspApi.query(
        new NspDomain().name("test.domain.com"),
        new NspEndpoint()
            .name("test_ep").resourcePath("/sensor/temperature")
            .method(NspMethod.SUBSCRIBE)
);
```

As a result we get an NspResult object. Only the status is provided.

```
if (nspResult.status() == Nsp.STATUS_OK) {
        System.out.println("Subscription added");
}
```

8.7.2 Unsubscribe from a resource

Prerequisites: To unsubscribe from a resource, you must know the domain, endpoint and resource to which you want to subscribe.

The NSP Web interface will receive the event request:

```
DELETE /events/subscription/resource/test_ep.test.domain.com/sensor/temperature
```

Remove the subscription from a resource by calling the query method in the NSP API. Give a name for the domain and the endpoint and path for the resource. Set the endpoint's method as `UNSUBSCRIBE`.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint()
        .name("test_ep")
        .resourcePath("/sensor/temperature")
        .method(NspMethod.UNSUBSCRIBE)
);
```

As a result we get an NspResult object. Only the status is provided.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Subscription deleted");
}
```

8.7.3 Check subscription to a resource

Prerequisites: To check if the subscription to a resource exists, you must know the domain, endpoint and resource to which you want to subscribe.

The NSP Web interface will receive the event request:

```
GET /events/subscription/resource/test_ep.test.domain.com/sensor/temperature
```

Check the subscriptions to a resource by calling the `query()` method in the NSP API. Give name for the domain and the endpoint and path for the resource. Set the endpoint's method as `SUBSCRIPTIONS`.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint()
        .name("test_ep")
        .resourcePath("/sensor/temperature")
        .method(NspMethod.SUBSCRIPTIONS)
);
```

As a result we get an NspResult object. Only the status is provided.

If a subscription to the resource is found, `STATUS_OK` is returned. If not, then `STATUS_NO_CONTENT` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Subscription found");
}
if (nspResult.status() == Nsp.STATUS_NO_CONTENT) {
    System.out.println("Subscription not found");
}
```

8.7.4 Subscribe to an endpoint

Prerequisites: To subscribe to an endpoint, you must know the domain and endpoint to which you want to subscribe.

The NSP Web interface will receive the event request:

```
PUT /events/subscription/endpoint/test_ep.test.domain.com
```

Add the subscription to an endpoint by calling the `query()` method in the NSP API. Give a name for the domain and the endpoint. Set the endpoint's method as `SUBSCRIBE`.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint().name("test_ep").method(NspMethod.SUBSCRIBE)
);
```

As a result we get an NspResult object. Only the status is provided.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Subscription added");
}
```

8.7.5 Unsubscribe from an endpoint

Prerequisites: To unsubscribe from an endpoint, you must know the domain and endpoint to which you want to subscribe.

The NSP Web interface will receive event uri:

```
DELETE /events/subscription/endpoint/test_ep.test.domain.com
```

Add the subscription to an endpoint by calling the `query()` method in the NSP API. Give name for the domain and the endpoint. Set the endpoint's method as `UNSUBSCRIBE`.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com"),
    new NspEndpoint().name("test_ep").method(NspMethod.UNSUBSCRIBE)
);
```

As a result we get an NspResult object. Only the status is provided.

```
 if (nspResult.status() == Nsp.STATUS_OK) {System.out.println("Subscription deleted");}
```

8.7.6 Check subscription to an endpoint

Prerequisites: To check if the subscription to an endpoint exists, you must know the domain and endpoint to which you want to subscribe.

The NSP Web interface will receive event uri:

```
GET /events/subscription/endpoint/test_ep.test.domain.com
```

Check the subscriptions to a resource by calling the `query()` method in the NSP API. Give a name for the domain and the endpoint and path for the resource. Set the endpoint's method as SUBSCRIPTIONS.

```
NspResult nspResult = nspApi.query(
    new NspDomain()
        .name("test.domain.com"),
    new NspEndpoint()
        .name("test_ep")
        .method(NspMethod.SUBSCRIPTIONS)
);
```

As a result we get an NspResult object. Only the status is provided.

If a subscription to a resource is found, `STATUS_OK` is returned. If not, then `STATUS_NO_CONTENT` is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Subscription found");
}
if (nspResult.status() == Nsp.STATUS_NO_CONTENT) {
System.out.println("Subscription not found");
}
```

8.7.7 Get notifications

There are two ways to get notifications from NSP, long polling and using notification servlet.

Prerequisites: You can get notifications from a whole domain. To get notifications, you must know the name of the domain. You must also implement a handler for incoming notifications to your application. The handler implementation must implement the functionality required by your application to handle incoming notifications. Also, started web application container has binded nsp notification servlet to handle incoming http messages.

The Java SDK is *listening* for NSP notifications and passes the incoming notifications to all handlers registered for the domain.

With long polling:

The NSP Web interface will receive the event request:

```
GET /events/observe/test.domain.com
```

With subscription server:

The NSP Web interface will receive the request:

```
GET /events/subscriptionserver/test.domain.com?uri=[protocol]://[host]:[port]/[path]
```

Host, port and path are configurable in `nsp.properties`. After that NSP will post notifications to that uri.

Execute the query by calling the query -method in the NSP API. Give the domain name and method `OBSERVE`.

Add the instance of your notification handler implementation class to the query.

```
NspResult nspResult = nspApi.query(
    new NspDomain().name("test.domain.com").method(NspMethod.OBSERVE),
    new MyNotificationHndlr()
);

// here is the application's implementation of the notification handler
public class MyNotificationHndlr implements NspNotificationHandler {
     @Override
     public void handle(NspNotification notification) {
        System.out.println("Notification received");
     }
 }
```

As a result we get an NspResult object. If the notification handler was added successfully, *STATUS_OK* is returned.

```
if (nspResult.status() == Nsp.STATUS_OK) {
    System.out.println("Ready to receive notifications from NSP");
}
```

Chapter 9
Quick Start Guide for the Lighting Example

This chapter describes the installation, configuration, and interfaces of the NanoService Lighting example.

It comprises the following sections:

9.1 Installation

The requirements for the NSP Lighting example are:

- Oracle Java JVM v1.7.
- NSP 1.9.

9.1.1 Configuration

A default configuration can be used when running on the same server with the NSP.

All configuration files are placed in the folder:

`<lighting-path>/conf/*`

NSP

Settings for the NSP connections are stored in:

`<lighting-path>/conf/nsp.properties`

Table 9-1 lists the frequently used NSP properties:

Table 9-1 NSP properties

Parameter	Description
NSP_DOMAIN	NSP address
NSP_PORT	NSP port
NSP_USER	NSP user name
NSP_SECRET	NSP password

Core

The main configuration file for the core properties is in:

`<lighting-path>/conf/core.properties`

Table 9-2 lists the main sections.

Table 9-2 Core properties

Section	Description
General	General parameters, such as application name, data path, and menu items
Historical storage	Enables or disables storing historical data for selected resources
Simulator	Internal simulator (when in use no connection to NSP is established)
Map	Map component specific configuration (such as default center point and manual range)
Event	Email setting for events
Geocoding	Enables or disables using geolocation lookups

All properties are commented, for details, see `core.properties`.

Profiles

The profile filename must conform to the pattern:

`<lighting-path>/conf/*profile.json`

Each profile configuration defines resource semantics with:

Name	A user-friendly name that is shown instead of a uri-path.
Commands	Operation that you can do on resource. There are two types of commands: • Normal, with a predefined message body to send. • Free text, where you will be asked to type a message before sending.
`State`	State that is added to an endpoint if the defined conditions are met.

```
/*

Semantics for temperature resource that will add alarm state 'cold' if
temperature goes below 10

*/
{
   "name": "Temperature",
   "uri-path": "/dev/V",
   "state":[
       { "name":"cold", "less":10, "status":"ALARM" }
   ]
}
```

User accounts

A list with user accounts and their permissions is defined in:

`<lighting-path>/conf/user.properties`

The properties file should contain entries in the form:

```
username=sha-password,grantedAuthority[,grantedAuthority][,enabled|disabled]
```

Where:

Password	To generate a password, you can use any sha generator tool, for example: `http://hash.online-convert.com/sha1-generator`
Roles	The available roles (authorities) are listed in Table 9-3.

Table 9-3 Authorities

Role	Description
`ROLE_ADMIN`	Full permission
`ROLE_USER`	Permission for map and items
`ROLE_EVENTS`	Permission for events
`ROLE_SCHEDULE`	Permission for schedule
`ROLE_NODE_NW_OPERATOR`	Permission for operations on /nw function set

For example, if the username is `demo` and the password is `demo`:

```
demo=ieSV55Qc+eQOaYDRSha/AjzNTJE=,ROLE_USER,enabled
```

Functional set

Mapping a functional set name is defined in the file:

`<lighting-path>/conf/functional-set.json`

The `json` file defines a user-friendly name for the functional set.

Example:

```
[
    {
        "path": "/dev",
        "name": "Device"
    }
]
```

Icons

Icons are defined in the file:

`<lighting-path>/conf/icons.json`

The file defines a list of icons that can be assigned to an endpoint in the Endpoint details view. Table 9-4 and Table 9-5 present the icon definitions.

Table 9-4 Icon definition

Name	Icon name
default-icon-url	Default icon url
endpoint-types (Optional)	An array of endpoint types that this icon is assigned to automatically
icons (Optional)	An array of icon-condition objects

Table 9-5 Icon condition definition

Name	Description
icon-url Icon	url
state	An array with states that this icon represents. Special state (#ALARM) covers all alarm states.

```
[
    /*
        Google default marker
    */
    {
        "name": "Google Marker",
        "default-icon-url": "http://maps.google.com/mapfiles/marker.png"
    },
    /*
        Sensor marker that is green, and if any alarm state then changes to red
    */
    {
        "name": "Sensor",
        "default-icon-url":"http://maps.google.com/mapfiles/marker_green.png",
                    "icons": [
                    {
                        "icon-url": "http://maps.google.com/mapfiles/marker.png",
                        "state": ["#ALARM"]
                    }
                ]
            }
        ]
```

9.2 Starting the Lighting example

This section describes how to run the Lighting example.

9.2.1 Linux:

Enter the following commands:

```
cd connected-home-{version}/bin
./runConnectedHome.sh
```

——— **Note** ———

Execution rights should already be set before running the example. If not, set the permissions with:

```
chmod +x runConnectedHome.sh
```

——— **Note** ———

If you are using a remote console, for example SSH connection, the application will be closed if the remote connection is lost. To prevent this, use the `nohup` command:

```
nohup ./runConnectedHome.sh
```

9.2.2 Windows:

Enter the following commands:

```
cd connected-home-{version}/bin
runConnectedHome.bat
```

9.3 User interface

This section describes the user interface of the Lighting example.

9.3.1 Login

To use the Lighting application, you must authenticate yourself:

1. In the login window, enter your username and password. See Figure 9-1.
2. Click **Login**.

You can find the user credentials in the `/conf/core.properties` file.

The default username is `demo` and the default password is `demo`.

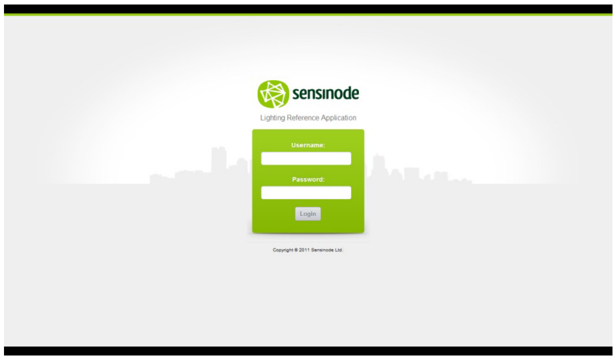

sensinode
Lighting Reference Application
Username:
Password:
Login
Copyright © 2011 Sensinode Ltd.

Figure 9-1 Login view

9.3.2 Map

The map component shows all available endpoints placed with coordinates (see Figure 9-2). Each endpoint is connected with its parent. The map shows the physical connection of a wireless sensor network. The connection quality (RSSI) is indicated by color, where:

- Green: very good signal.
- Red: bad signal.
- Black: unknown signal quality.

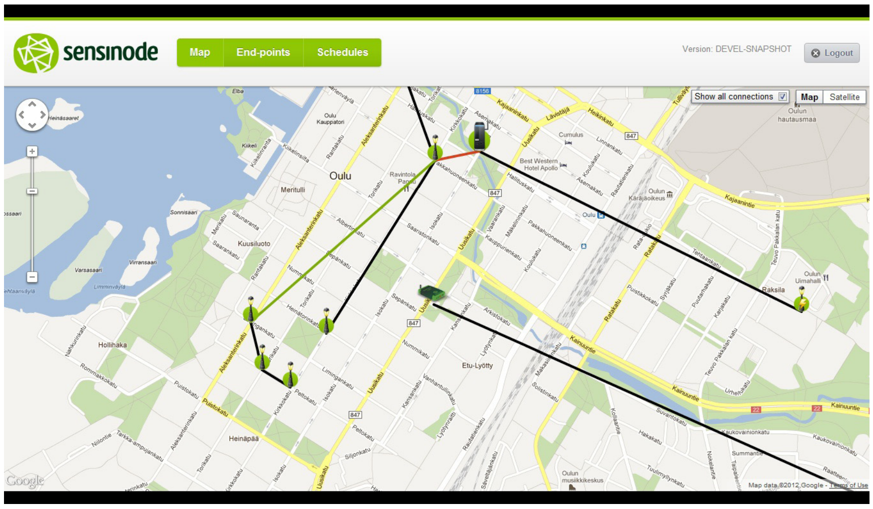


Figure 9-2 Map view

To correct an endpoint position on the map:

1. Enable the map manual positioning in the Node details view (see Figure 9-6 on page 9-9).
2. Drag the endpoint on the map to the correct position.

The Node details view presents all information about the selected endpoint. You can read the resource content. Some resources have also predefined actions that you can perform, for example, turn the light on (see Figure 9-7 on page 9-10). The map is constantly updated so you do not need to reload the page.

9.3.3 Creating a group on a map

You can make a selection of endpoints (see Figure 9-3). Use the context menu to:

- Create a new group.
- Add endpoints to an existing group.

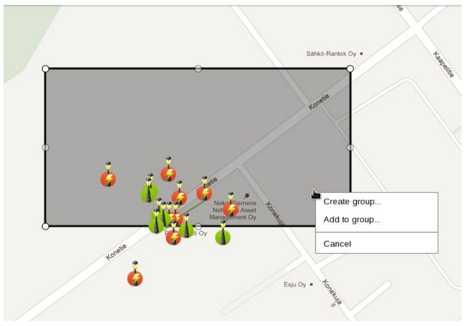


Figure 9-3 Endpoint selection on a map

9.3.4 Showing a group on a map

You can see the contents of a group in a map (see Figure 9-4). Endpoints that do not belong to a selected group, are shown as gray blobs. Use the selection and context menu to:

- Add endpoints to a selected group.
- Remove endpoints from a selected group.

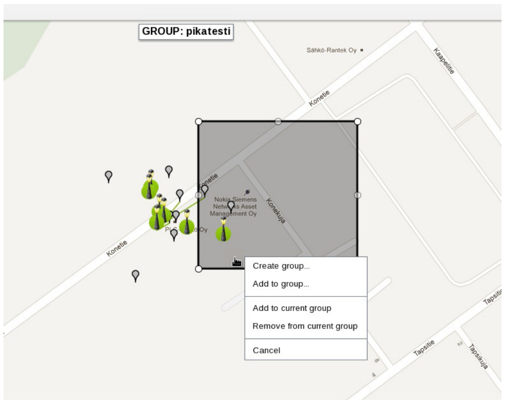


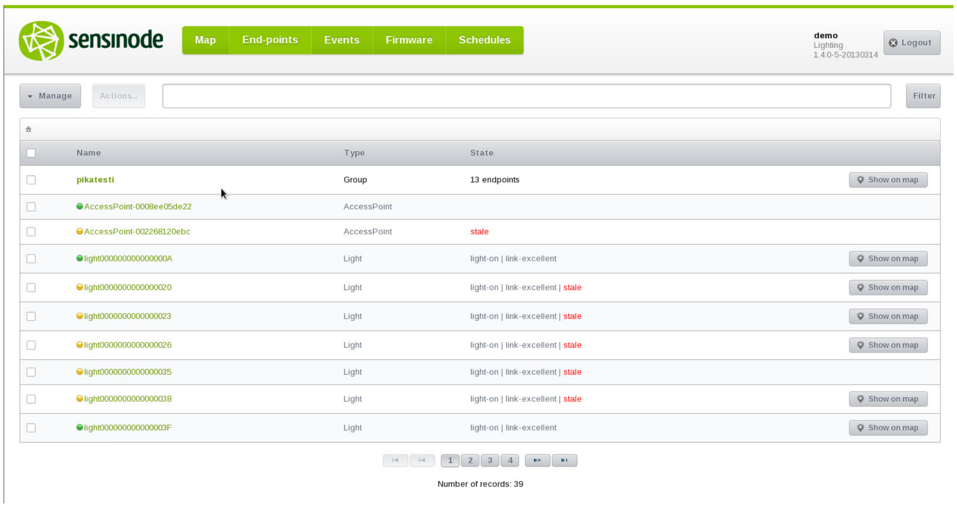
Figure 9-4 Show group

9.3.5 Endpoints

The Endpoints window (see Figure 9-5) shows all endpoints and groups. In this window, you can:

- Add a group.
- Remove a group.
- Add endpoints to a group.
- Remove endpoints from a group.

Use **Search** to find specific endpoints.

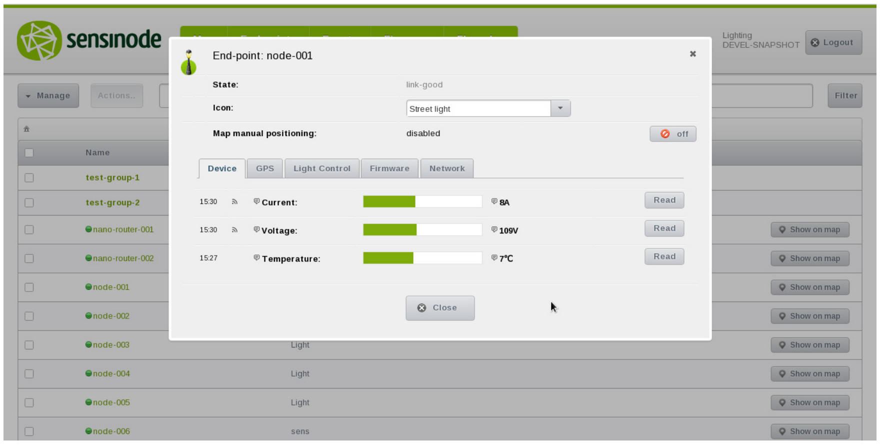


Name	Type	State	
pikatesti	Group	13 endpoints	Show on map
● AccessPoint-0008ee05de22	AccessPoint		
● AccessPoint-002268120ebc	AccessPoint	stale	
● light000000000000000A	Light	light-on | link-excellent	Show on map
● light0000000000000020	Light	light-on | link-excellent | stale	Show on map
● light0000000000000023	Light	light-on | link-excellent | stale	Show on map
● light0000000000000026	Light	light-on | link-excellent | stale	Show on map
● light0000000000000035	Light	light-on | link-excellent | stale	
● light0000000000000038	Light	light-on | link-excellent | stale	Show on map
● light000000000000003F	Light	light-on | link-excellent	Show on map

1 2 3 4

Number of records: 39

Figure 9-5 Endpoints view



End-point: node-001

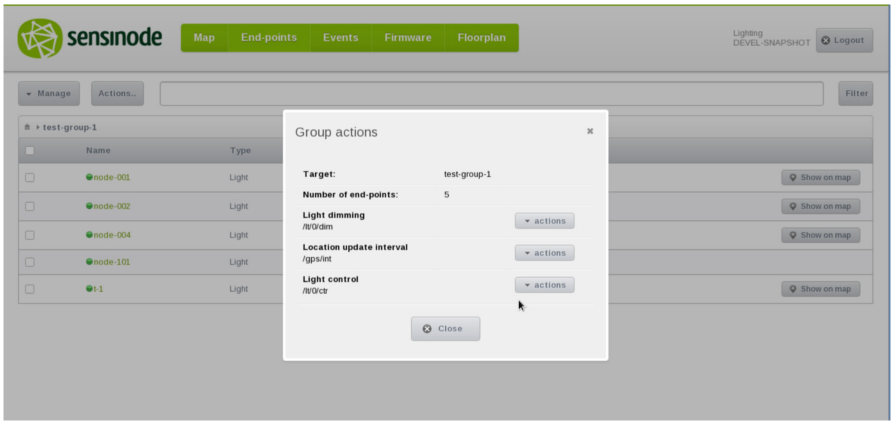
State:	link-good
Icon:	Street light
Map manual positioning:	disabled

off

Time	Measurement	Value
15:30	Current:	8A
15:30	Voltage:	109V
15:27	Temperature:	7°C

Name	Type
test-group-1	
test-group-2	
● nano-router-001	
● nano-router-002	
● node-001	
● node-002	
● node-003	Light
● node-004	Light
● node-005	Light
● node-006	sens

Figure 9-6 Node details view



Name	Type
node-001	Light
node-002	Light
node-004	Light
node-101	Light
t-1	Light

Group actions

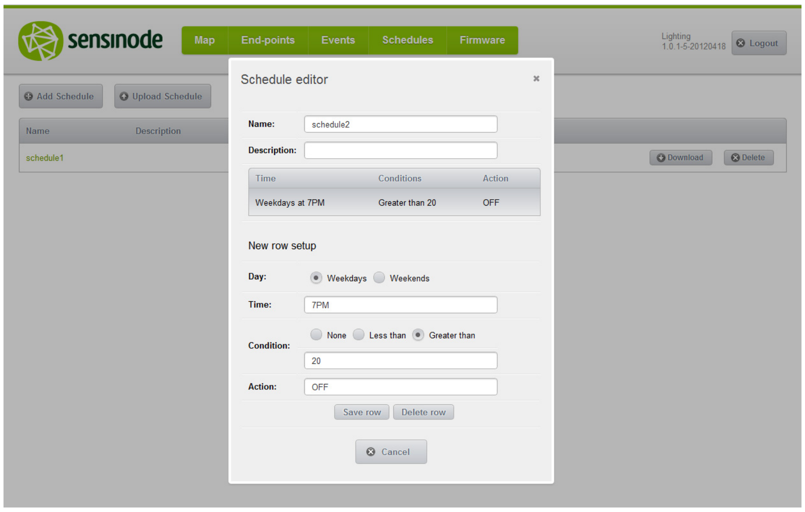
Target:	test-group-1
Number of end-points:	5
Light dimming /lt/0/dim	actions
Location update interval /gps/int	actions
Light control /lt/0/ctr	actions

Figure 9-7 Group actions

9.3.6 Schedules

The Schedules window manages schedules (see Figure 9-8). In this window, you can:

- Create a schedule with a web form.
- Upload a schedule from an existing file.



Name	Description
schedule1	

Schedule editor

Name: schedule2

Description:

Time	Conditions	Action
Weekdays at 7PM	Greater than 20	OFF

New row setup

Day: [✓ Weekdays] Weekends

Time: 7PM

Condition: None Less than [✓ Greater than]
20

Action: OFF

Figure 9-8 Schedules view

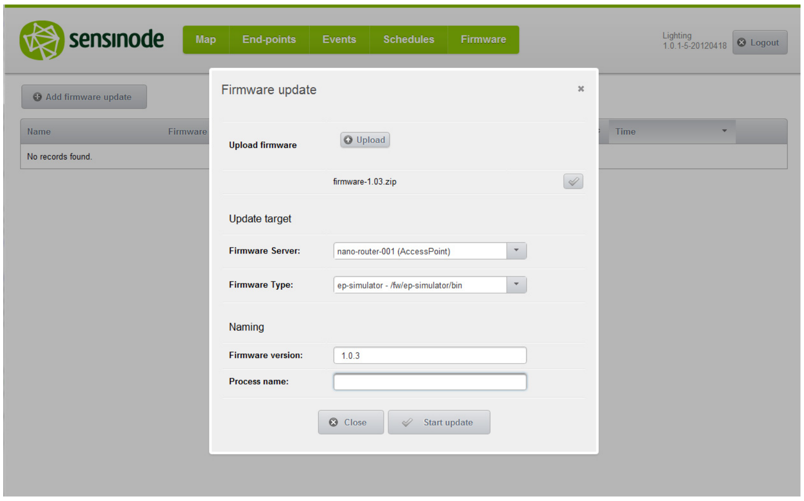
9.4 Firmware updates

Caution

Firmware update must be done with extra caution!

To update the sensor or access point firmware, use the Firmware update component:

1. In the Firmware update window (see Figure 9-9), select:
   - Firmware binaries.
   - The destination firmware server.
   - Firmware type.
   - Firmware version.
2. Click **Start update** to upload the binaries to the firmware server.
3. After the upload is finished, the firmware list is updated (see Figure 9-10).
4. Now you can enable the endpoints in the Node details view (see Figure 9-6 on page 9-9) or in the Group actions view (see Figure 9-7 on page 9-10) to download the firmware from the firmware server.



Firmware update

Upload firmware: firmware-1.03.zip

Update target

Firmware Server: nano-router-001 (AccessPoint)

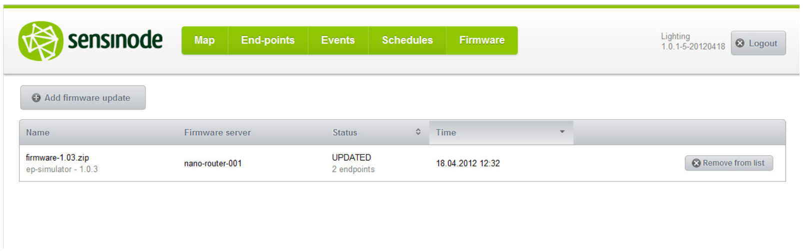
Firmware Type: ep-simulator - /fw/ep-simulator/bin

Naming

Firmware version: 1.0.3

Process name:

Figure 9-9 Firmware update creator



Name	Firmware server	Status	Time
firmware-1.03.zip ep-simulator - 1.0.3	nano-router-001	UPDATED 2 endpoints	18.04.2012 12:32

Figure 9-10 Firmware update list

9.5 NanoService Semantic Set

The application uses NanoService Semantic Set to recognize sensor resources. Every resource describes the metadata in Table 9-6. The table also shows example resource values, such as temperature (in Celsius units).

Table 9-6 Temperature resources

Resource	Description
Uri path	`/dev/temp`
Resource type	`ucum:Cel`
Content type	`text/plain`
Interface description	`ns.wadl#s`
Content value	`"21" 0x3231`

9.5.1 Resource types

The Resource Type defines the semantic meaning of the resource, and may reference an ontology with an XML namespace style prefix (such as `ns:`). Application-specific resource types do not use a known prefix. This semantic set supports the following ontologies:

Table 9-7 Semantic set

Type	Ontology	Reference
ns:	NanoService Resource Ontology	Table 9-9 on page 9-13
ucum:	Unified Code for Units of Measure (c/s)	`http://unitsofmeasure.org`

Basic sensor resource types can be notated using *The Unified Code for Units and Measurement* [`http://aurora.regenstrief.org/~ucum/ucum.html`] with the prefix "`ucum:`" and the c/s format of the unit or measurement. See Table 9-8.

Table 9-8 Resource types

Type	Description	Unit	Notes	Examples
ucum:Cel	Temperature in Celsius	°C	signed integer	'27', '-7'
ucum:degF	Temperature in Fahrenheit	°F	signed integer	'104', '-1'
ucum:W	Power	W	unsigned integer	'84', '0'
ucum:lux	Illuminance	lx	unsigned integer	'267'
ucum:sec	Time in seconds	sec	integer	'54'
ucum:min	Time in minutes	min	integer	'54'
ucum:db	Decibel	db	integer	'6'

Non-physical measurement related resources are defined using Sensinode's own NanoService ontology. See Table 9-9.

Table 9-9 Measurement related resources

Type	Description	Notes	Examples
ns:relay	Power relay that has a boolean state	values: 1, 0	-
ns:switch	A switch that has a boolean state	values: 1, 0	-
ns:boolean	A generic digital input sensor	values: 1, 0	-
ns:motion-count	A motion sensor that increases per motion event	integer	-
ns:image	A still image	-	-
ns:gpsloc	Latitude and longitude information	[-180 - +180], [¬90 - +90]	'65.032887,25.506477', '-22.917923,-43.286133'
ns:range	Range of integer values	-	-
ns:v6addr	An IPv6 address in alphanumeric hex format	-	2001:1234::FADE
ns:euid64	A 64-bit EUID in alphanumeric hex format	-	FE:ED:FA:DE:80:88:77:70
ns:counter	A generic positive integer counter resource	-	10001
ns:rssi	RF received signal strength indicator value	-	-78
ns:fw	Firmware	-	-
ns:devtype	Device type	-	-
ns:dom	NSP domain	-	-

Chapter 10
Quick Start Guide for the Connected Home Example

This chapter describes the installation, configuration, and interfaces of the NanoService Connected Home example.

It comprises the following sections:

10.1 Installation

The requirements for the NSP Connected Home example are:

- Oracle Java JVM v1.7.
- NSP 1.9.

10.1.1 Configuration

A default configuration can be used when running on the same server with NSP 1.9. Use the file `/conf/nsp.properties` to change the NSP connection settings. The NSP configuration properties are listed in Table 10-1.

Table 10-1 Configuration properties

Parameter	Description
NSP_DOMAIN	NSP address
NSP_PORT	NSP port
NSP_USER	NSP user name
NSP_SECRET	NSP password

The file `/conf/core.properties` contains the relevant properties listed in Table 10-2.

Table 10-2 Core properties

Property	Description
core.nsp-domain	The name of a domain that uses endpoints from the core.
core.data-path	Defines the data path where the application data is stored.
core.username	A username for logging in an application (using a browser)
core.password	A password for logging in an application (using a browser)
core.observation-request-interval	Observation request interval in milliseconds. Prevents sending too many requests at once.
core.name	Defines the name of this application.
core.extra-components	Defines a list with extra components divided by a comma.
SIMULATOR	
core.nanoservice	A connector for NSP connection, defines if using a simulator or a connection to a real NSP.
simulator.number-of-simulated-nodes	The number of simulated nodes
simulator.notification-frequency	Notification frequency in milliseconds, which is how often the resource changes are simulated.
simulator.max-response-delay	Simulated response delay for resource requests in milliseconds
EVENT	
event.smtp.enabled	Defines if the SMTP is enabled.

Table 10-2 Core properties (continued)

Property	Description
event.smtp.sender	SMTP sender email address
event.smtp.host event.smtp.port event.smtp.secure	SMTP server host, note that ip address instead of host
event.smtp.use-authentication	If SMTP server requires authentication
event.smtp.username	SMTP username
event.smtp.password	SMTP password
GEOCODING	
geocoding.inuse	Defines if the reverse geocoding of endpoint addresses is in use.
geocoding.class	Fully qualified class name of geolocator implementation, extending class GeoLocator
geocoding.min-interval	Interval of geocoding requests to the server, in milliseconds. The recommended value depends highly on the implementation used.
geocoding.contact	Your contact information. Some implementations (for example, openstreetmap) send contact email information to service in case they need to contact you.
geocoding.licence-key	Licence key for service. Some implementations might need a licence key for service requests.

10.2 Starting the Connected Home example

This section describes how to run the Connected Home example.

10.2.1 Linux:

Enter the following commands:

```
cd connected-home-{version}/bin
./runConnectedHome.sh
```

——— **Note** ———

Execution rights should already be set before running the example. If not, set the permissions with:

```
chmod +x runConnectedHome.sh
```

———————

——— **Note** ———

If you are using a remote console, for example SSH connection, the application will be closed if the remote connection is lost. To prevent this, use the `nohup` command:

```
nohup ./runConnectedHome.sh
```

———————

10.2.2 Windows:

Enter the following commands:

```
cd connected-home-{version}/bin
runConnectedHome.bat
```

10.3 User interface

This section describes the user interface of the Connected Home example.

10.3.1 Login

To use the Connected Home application, you must authenticate yourself:

1. In the login window, enter your username and password. See Figure 10-1.
2. Click **Login**.

You can find the user credentials in the `/conf/core.properties` file.

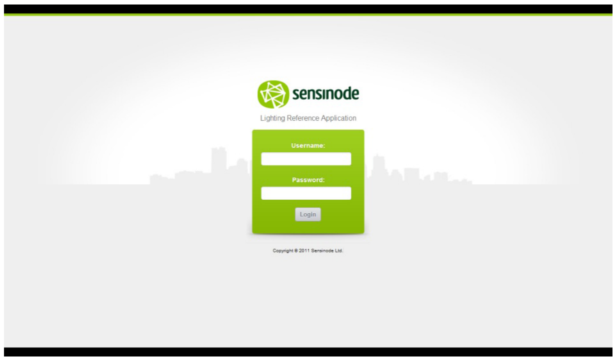

sensinode
Lighting Reference Application
Username:
Password:
Login
Copyright © 2011 Sensinode Ltd.

Figure 10-1 Login view

10.3.2 Floorplan

The Floorplan window (see Figure 10-2 on page 10-6) shows floorplans and the endpoints in it. In this window, you can:

- Create a new floorplan with a custom image and name.
- Edit an existing floorplan.
- Delete a floorplan.
- Relocate an endpoint in the floorplan.
- Remove an endpoint from the floorplan.

To relocate an endpoint in the floorplan, drag it in the correct place.

Click an endpoint to open a dialog where you can remove the endpoint from the floorplan.

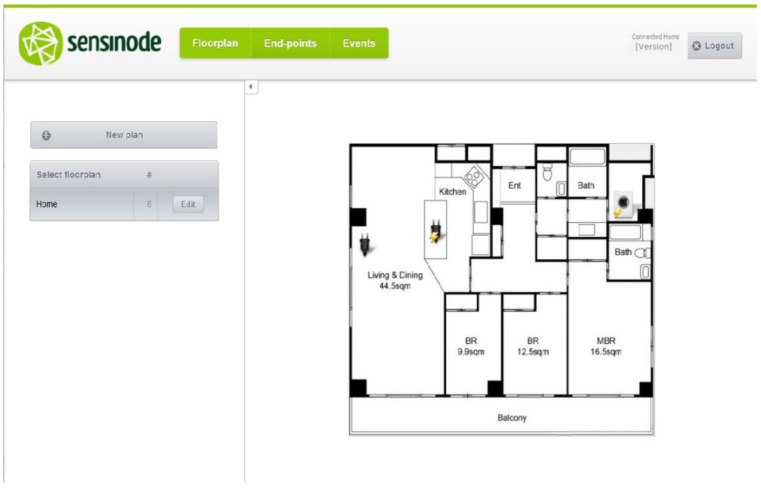
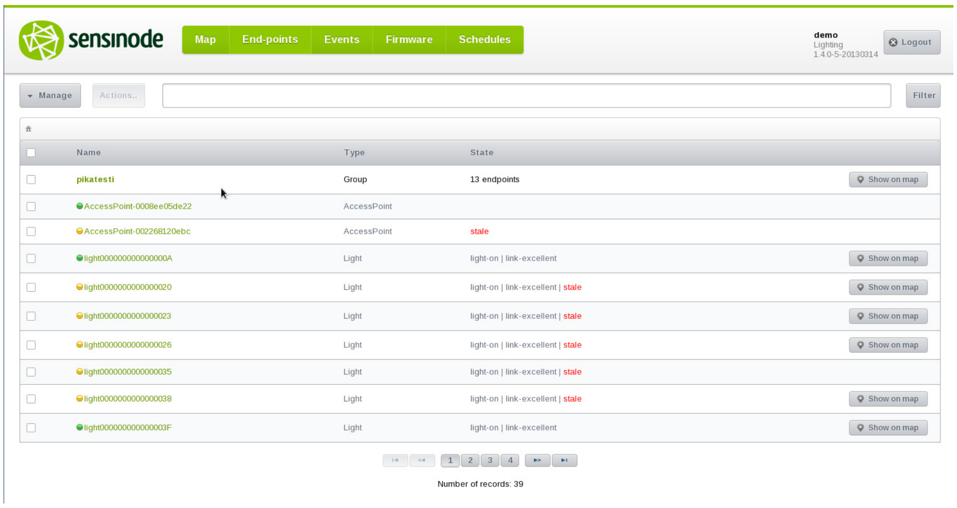


Figure 10-2 Floorplan

10.3.3 Endpoints

The Endpoints window (see Figure 10-3) shows all endpoints and groups. In this window, you can select endpoints and add them to an existing floorplan. Use **Search** to find specific endpoints.



	Name	Type	State	
☐	pikatesti	Group	13 endpoints	Show on map
☐	AccessPoint-0008ee05de22	AccessPoint		
☐	AccessPoint-002268120ebc	AccessPoint	stale	
☐	light000000000000000A	Light	light-on | link-excellent	Show on map
☐	light0000000000000020	Light	light-on | link-excellent | stale	Show on map
☐	light0000000000000023	Light	light-on | link-excellent | stale	Show on map
☐	light0000000000000026	Light	light-on | link-excellent | stale	Show on map
☐	light0000000000000035	Light	light-on | link-excellent | stale	
☐	light0000000000000038	Light	light-on | link-excellent | stale	Show on map
☐	light000000000000003F	Light	light-on | link-excellent	Show on map

1 2 3 4

Number of records: 39

Figure 10-3 Endpoints view

10.4 NanoService Semantic Set

The application uses NanoService Semantic Set to recognize sensor resources. Every resource describes the metadata in Table 10-3. The table also shows example resource values, such as temperature (in Celsius units).

Table 10-3 Resource and metadata

Resource	Description
Uri path	`/dev/temp`
Resource type	`ucum:Cel`
Content type	`text/plain`
Interface description	`ns.wadl#s`
Content value	`"21" 0x3231`

10.4.1 Resource types

The Resource Type defines the semantic meaning of the resource, and may reference an ontology with an XML namespace style prefix (such as `ns:`). Application-specific resource types do not use a known prefix.

The semantic set supports the ontologies presented in Table 10-4.

Table 10-4 Semantic set

Type	Ontology	Reference
ns:	NanoService Resource Ontology	Table 10-6 on page 10-8
ucum:	Unified Code for Units of Measure (c/s)	`http://unitsofmeasure.org`

Basic sensor resource types can be notated using *The Unified Code for Units and Measurement* [`http://aurora.regenstrief.org/~ucum/ucum.html`] with the prefix "`ucum:`" and the c/s format of the unit or measurement. See Table 10-5.

Table 10-5 Resource types

Type	Description	Unit	Notes	Examples
ucum:Cel	Temperature in Celsius	°C	signed integer	'27', '-7'
ucum:degF	Temperature in Fahrenheit	°F	signed integer	'104', '-1'
ucum:W	Power	W	unsigned integer	'84', '0'
ucum:lux	Illuminance	lx	unsigned integer	'267'
ucum:sec	Time in seconds	sec	integer	'54'
ucum:min	Time in minutes	min	integer	'54'
ucum:db	Decibel	db	integer	'6'

Non-physical measurement related resources are defined using Sensinode's own NanoService ontology. See Table 10-6.

Table 10-6 Measurement related resources

Type	Description	Notes	Examples
ns:relay	Power relay that has a boolean state	values: 1, 0	-
ns:switch	A switch that has a boolean state	values: 1, 0	-
ns:boolean	A generic digital input sensor	values: 1, 0	-
ns:motion-count	A motion sensor that increases per motion event	integer	-
ns:image	A still image	-	-
ns:gpsloc	Latitude and longitude information	[-180 - +180], [¬90 - +90]	'65.032887,25.506477', '-22.917923,-43.286133'
ns:range	Range of integer values	-	-
ns:v6addr	An IPv6 address in alphanumeric hex format	-	2001:1234::FADE
ns:euid64	A 64-bit EUID in alphanumeric hex format	-	FE:ED:FA:DE:80:88:77:70
ns:counter	A generic positive integer counter resource	-	10001
ns:rssi	RF received signal strength indicator value	-	-78
ns:fw	Firmware	-	-
ns:devtype	Device type	-	-
ns:dom	NSP domain	-	-

10.4.2 Interface descriptions

Table 10-7 lists the interfaces and associated descriptions and methods.

Table 10-7 Interface descriptions

Type	Interface description	CoAP methods
Sensor	ns.wadl#s	GET
Parameter	ns.wadl#p	GET, PUT
Actuator	ns.wadl#a	GET, PUT
Collection	ns.wadl#c	GET

For more information on the interface, see:

`http://datatracker.ietf.org/doc/draft-ietf-core-link-format/`

10.4.3 Configuration of resource semantics

Resource semantics are mapped in file `sensinode-profile.json` in `/conf` folder. Also files in the folder ending with a name `*profile.json` are interpreted as semantics files. These resource semantics are then used in the UI to rename resources and their values with user-readable content, defining possible alarm values and enabling the possibility to change the resource value in devices.