Nymi API for Linux Guide

Nymi Connected Worker Platform

3.0

2021-05-03

Contents

Preface.......... 4

Nymi API for Linux Overview.......... 6

Nymi API for Linux Architecture.......... 6
Development Tools.......... 7
Supported Platforms.......... 7
Supported NFC Reader.......... 7
Nymi API for Linux Sample Application.......... 7

Nymi API for Linux Installation.......... 8

Installing the Nymi API for Linux.......... 8
Importing TLS Certificates Obtained from the NES Server.......... 9
Copying and Extracting the TLS Certificate.......... 9

Configuring NFC readers.......... 10

Nymi Component Configuration.......... 11

Uninstalling Nymi Linux Runtime.......... 12

Creating NEAs with `Nymi API`.......... 13

Call Concurrency.......... 13
Request and Response.......... 13

Request Operations.......... 15

subscribe operation.......... 17
init operation.......... 18
lookup.......... 22
assert_identity.......... 24
presence update.......... 26

Response Messages and Notifications.......... 28

update function.......... 29

assert_identity response....30
initialization error notifications....31
presence notifications....31
Bluetooth notifications....34
Intent Notification....34
Editing the nbe.toml File....35

Error Handling....38
Status Code....38

Troubleshooting....41
Enable debugging....41
Service information and runtime errors....41
Checking the service status....42
Starting services commands....42

Appendix....43
Authentication requirements....43
Acquire an Authentication Token....43

Preface

Nymi™ provides periodic revisions to the Nymi Connected Worker Platform. Therefore, some functionality that is described in this document might not apply to all currently supported Nymi products. The product release notes provide the most up to date information.

Purpose

This document is part of the `Connected Worker Platform` (CWP) documentation suite.

This document provides information about how to use the functionality that is available in the `NAPI` that is part of the Connected Worker Platform.

Audience

This guide provides information to Developers.

Revision history

The following table outlines the revision history for this document.

Table 1: Revision history

Version	Date	Revision history
03	May 3, 2021	Update to reflect Nymi Enterprise Edition rebrand to Connected Worker Platform.
02	April 15, 2020	This guide is reissued due to document version update. There are no content changes since NEE 2.6.0.
01	September 18, 2020	Reissued to clarify behaviour for the presence operation, presence notification and bluetooth notifications.

Related documentation

- **Nymi Connected Worker Platform NES Deployment Guide**

  This document provides an overview of the components and the steps that are required to deploy the Nymi Enterprise Server (NES). This installation uses the `Nymi Token Service` to install certificates that enable communication between components. This document also provides information about deploying the Connected Worker Platform in a Citrix or RDP environment.

- **Nymi Connected Worker Platform Administration Guide**

  This document provides information about how to use the `NES Administrator Console` to manage the `Connected Worker Platform` (CWP) system. This document describes how to set up, use and manage the Nymi Band™, and how to use the `Nymi Band Application`. This document also provides instructions on deploying the Nymi Band Application and Nymi Runtime components.
- **Connected Worker Platform Release Notes**

  This document provides supplemental information about the Connected Worker Platform, including new features, limitations, and known issues with the Connected Worker Platform components.
- **Nymi Connected Worker Platform Troubleshooting Guide**

  This document provides information about how to troubleshoot issues and the error messages that you might experience with the `NES Administrator Console`, the Nymi Enterprise Server deployment, the Nymi Band, and the `Nymi Band Application`.

How to get product help

If the Nymi software or hardware does not function as described in this document, contact your administrator for immediate support. Alternatively, you can submit a support ticket to Nymi, or email support@nymi.com

How to provide documentation feedback

Feedback helps Nymi to improve the accuracy, organization, and overall quality of the documentation suite. You can submit feedback by using support@nymi.com

Nymi API for Linux Overview

The Nymi SDK provides developers with libraries, APIs, sample code and documentation for building a Nymi-enabled Application (NEA). The `Nymi API for Linux` architecture is part of the Nymi SDK.

This guide provides information about how to develop a Nymi-enabled Application using the `Nymi API for Linux`.

The Nymi SDK contains components that enable you to build Nymi-enabled Application (NEA):

- `Nymi Runtime`:provides developers with tools that enable them to create a Nymi-enabled Application. The Nymi SDK and Runtime handle the primary functions of the Nymi Band communication. The Nymi SDK consists of components that handle the business logic necessary in delivering functionality.
- `Nymi Agent`: provides BLE management, manages operations and message routing. Facilitates communication between NEAs and the Nymi Band, and maintains knowledge of the Nymi Band presence and authenticated states.
- `Nymi Bluetooth Endpoint Daemon (NBEd)`: host-side software that interfaces between the Bluegiga Dongle (BLE) and the Nymi Agent.
- `Nymi Agent Daemon (NymiAgentd)`: provides BLE management, manages operations and message routing.
- `Nymi API`: provides NEAs access to Nymi Band functionality using a dynamic link library. It also manages NEA certificates and allows secure communication with Nymi Bands using the Nymi Security Protocol. The `Nymi API` handle business logic through the operations, which send and receive request and response messages. `Nymi API`enable developers to create a Nymi-enabled Application. They connect the web, allowing developers, applications, and sites to tap into databases and services (or, assets)—much like open-source software. `Nymi API` does this by acting like a universal converter plug offering a standard set of instructions.

Nymi API for Linux Architecture

`Nymi API` supports enterprise integration and deployment of features that are available in the `Connected Worker Platform`.

Nymi has created APIs (referred to as NAPI) that supports the enterprise integration and development of features that are available in the `Connected Worker Platform`.

APIs enable developers to create Nymi-enabled Applications - They connect the web, allowing developers, applications, and sites to tap into databases and services (or, assets)—much like open-source software. APIs do this by acting like a universal converter offering a standard set of instructions.

For information about deploying in a Citrix or RDP environment, see the Nymi Connected Worker Platform NES Deployment Guide.

Development Tools

To develop NEAs on Linux platforms, you can use the following tools.

- Python 3
- any software language that supports C library, is supported for API development

Supported Platforms

`Nymi API for Linux` supports the following platforms.

- Linux CentOS 7, 64-bit

Supported NFC Reader

The `Nymi API for Linux` supports multiple NFC readers. A list of supported NFC readers is found in the Hardware requirements section of the Nymi Connected Worker Platform Administration Guide

The `Nymi Bluetooth Endpoint` monitors all attached and supported NFC readers and forwards events from all NFC readers (there is no preference between readers).

The NFC reader is connected to the user's terminal where the `Nymi Bluetooth Endpoint` is installed and are automatically detected by the `Nymi Bluetooth Endpoint`.

Nymi API for Linux Sample Application

Nymi offers you a sample application that demonstrates some of the functionality of the `Nymi API for Linux`. The sample application is written in Python language.

The sample application is located within the package at: *nymi-sdk/linux/examples/python*.

Nymi API for Linux Installation

The `Nymi API for Linux` development package contains the following components.

- *libnymi_api.so* : a library that exposes the Nymi functionality for both developers an Nymi-enabled Application
- *nymiagentd-x.x.x-x.x86_64.rpm* : Nymi Agent Daemon file
- *nbed-x.x.x-x.x86_64.rpm* : Nymi Bluetooth Endpoint Daemon file
- sample application and readme file

The RPM installation package is included in the Nymi release package. The following service are included in the installation package accessing the interface or BLE dongle.

- `Nymi Bluetooth Endpoint Daemon (NBEd)` accesses the serial port. Establishes and secures the Bluetooth connection to the Nymi Band.
- `Nymi Agent Daemon (NymiAgentd)` delegates requests and responses between the NEA and the Nymi Band.

Installing the Nymi API for Linux

The Linux runtime services, which include the `Nymi Bluetooth Endpoint Daemon (NBEd)` and `Nymi Agent Daemon (NymiAgentd)` are included in installation packages.

Nymi recommends that you install the `Nymi API for Linux` using the Linux YUM package manager commands. Using rpm may cause upgrade or uninstall issues.

Note: The `Nymi API for Linux` supports CentOS 7 only.

Note: You must have super user privileges (sudo) to install the files.

1. Obtain and download the Nymi Software package from your Nymi Solution Consultant.
2. In a terminal, within the software package, navigate to the following path: *nymi-sdk-5.4.0-xx/nymi-sdk/linux/RPMS/x86_64*
3. Extract the following rpm files:
   - *nbed-x.x.x-x.x86_64.rpm*
   - *nymiagentd-x.x.x-x.x86_64.rpm*

   where

   x.x.x-x represents the file version
4. Navigate to the location where you downloaded the nbed rpm file, and run the following command:

   ```
   $ sudo yum install ./<nbed-x.x.x-x.x86_64.rpm>
   ```

   where

   nbed-*x.x.x-x.x*86_64.rpm represents the file name.

5. Navigate to the location where you downloaded the nymiagentd rpm file, and run the following command:

```
$ sudo yum install ./<nymiagentd-x.x.x-x.x86_64.rpm>
```

   where

   nymiagentd-*x.x.x-x.x*86_64.rpm represents the file name.

Importing TLS Certificates Obtained from the NES Server

`Nymi API for Linux` supports TLS self-signed root CA certificates.

If your Nymi environment is already using self-signed root CA certificates (TLS server certificates) ensure that they are installed in the Trusted Root Certificate Store on the client, no further action is required.

If the TLS server certificate is missing from the trusted root certificate store, you can copy and extract the TLS server certificate that was issued by a Trusted Certificate Authority (CA). If TLS signing certificate is not signed by a trusted CA, the signing certificate need to be imported into Trusted Root Certificate.

Note: The following procedure assumes that the TLS server certificate and the associated private key are packaged in the same file. Depending on how the private key for your certificate is generated, your procedure might differ.

Copying and Extracting the TLS Certificate

Importing TLS server certificates into the Trusted Root Certificate Store.

1. Open a shell prompt in CentOS Linux.
2. Copy the certificates from their current location to the following location: */etc/pki/ca-trust/source/anchors*

   The following step assumes that the TLS server certificate and the associated private key are packaged in the same file.
3. Run the following command: *sudo update-ca-trust extract*.

Configuring NFC readers

PCSC-lite is a cross-platform API for accessing smart card readers. It is a dependency for using NFC readers with the `Nymi API for Linux`.

1. Open a terminal and download and install pcsc-lite using the following command:

   ```
   sudo yum install pcsc-lite
   ```

2. Enable pcsc-lite using the following command:

   ```
   sudo systemctl enable pcscd
   ```

3. Start pcsc-lite using the following command:

   ```
   sudo systemctl start pcscd
   ```

Nymi Component Configuration

In order to create an environment that can utilize the services contained in the `Nymi API for Linux`, specify the location of the `Nymi Agent` so that the `Nymi Bluetooth Endpoint` can connect to it.

The `Nymi API for Linux` is installed on each RDP client. The`Nymi API for Linux` service on each RDP client communicates with the `Nymi Agent` service, which is installed on a separate host, on websocket port 9120.

1. Navigate to the location where the *nbe* file is installed.
2. Open the */usr/sbin/nbe.toml* file.
3. Update the location of the `Nymi Agent`
    - agent_url = 'ws://<FQDN>:9120/socket/websocket'
4. Optionally, you can set the location of the `Nymi Bluetooth Endpoint` by configuring the *endpoint_id* parameter.
    - endpoint_id = "<unique ID>"

    By configuring the *endpoint_id* parameter, you need to use the subscribe operation. For more information about the subscribe operation, see the *Request Operation* section in this guide.

Uninstalling Nymi Linux Runtime

When uninstalling the `Nymi API for Linux`, the Nymi Agent and Nymi Bluetooth Endpoint services must be uninstalled on an individual basis.

To uninstall the Nymi Agent, open a terminal and run the following command:

```
$ sudo yum remove nymiagentd
```

To uninstall the Nymi Bluetooth Endpoint, open a terminal and run the following command:

```
$ sudo yum remove nbed
```

Note: After uninstalling the services, the configuration file and log files for Nymi Bluetooth Endpoint service remains available. The Nymi Agent log files are deleted.

Creating NEAs with `Nymi API`

This section provides information about NEAs and the supported functions and operations that developers can use to create NEAs in programming languages, such as Linux.

Note: In this document, the use of device refers to the Nymi Band.

Call Concurrency

NAPI has two FIFO (First-In, First-Out) message queues.

- Device queues—One message queue exists for each Nymi Band. When NAPI receives a device-related message, NAPI dispatches the message to the appropriate device message queue, in the order that the message is received. NAPI might dispatch messages to a device before dispatching messages that have been queued longer, to another device.
- Non-device queue—One global message queue that stores messages that are not related to a device operation, for example, the response for an *init()* call. NAPI dispatches non-device related messages to the queue in the order that the messages are received.

Request and Response

The *request()* and *update()* calls are handled differently in memory.

- *request()*—NEA supplies the request message in a memory buffer. Before the call returns, NAPI creates a copy of the message.
- *update()*—After the function returns, NAPI expects the NEA to copy the response message out of the memory address provided by the *update()* call, before calling the *update()* function again.

Request Message Format

Every request message is a JSON object with the following key-value pairs:

```
{
    operation: <"operation name">,
    [exchange: <"user defined">],
    payload: {
        operation specific request fields
    }
}
```

Response Message Format

Response messages have the same format as request message plus an extra fields related to the status of the request:

```
{
   operation: <"same as request">
   exchange: <"same as request" or null>
   status: <integer >= 0>,
   payload {
      operation specific response fields
   }
   error: {
      error_description: <"general error description">,
      error_specifics: <"">
   }
}
```

Request Operations

When a request is made to `NAPI` it is wrapped in this JSON object. The response from `NAPI` is wrapped in the same object with some additional attributes defined.

A *request()* call submits a message to `Nymi API`. `Nymi API` performs the operation that is contained in the message. The result of the operation is a response message. Use the *update* function to retrieve the response message.

The declaration for the request operation is as follows:

```
int request(const char* request_obj)
```

where:

- *request_obj* is a JSON string that requests `Nymi API` to perform an operation. A *request_obj* contains one or more properties. Each property has a name and a value.

  Note: A *request_obj* must contain one *operation* property.
- *request* returns one of the following values:
  - 0 when `Nymi API` accepts the *request_obj* for processing
  - 1 when `Nymi API` is not initialized and this request is not an *init* request. The NEA must run the *init* call before `Nymi API` can accept any messages other than *init*.

  Note: The NEA must already be initialized.

The following diagram shows the *request()* call and message handling workflow for device operations.

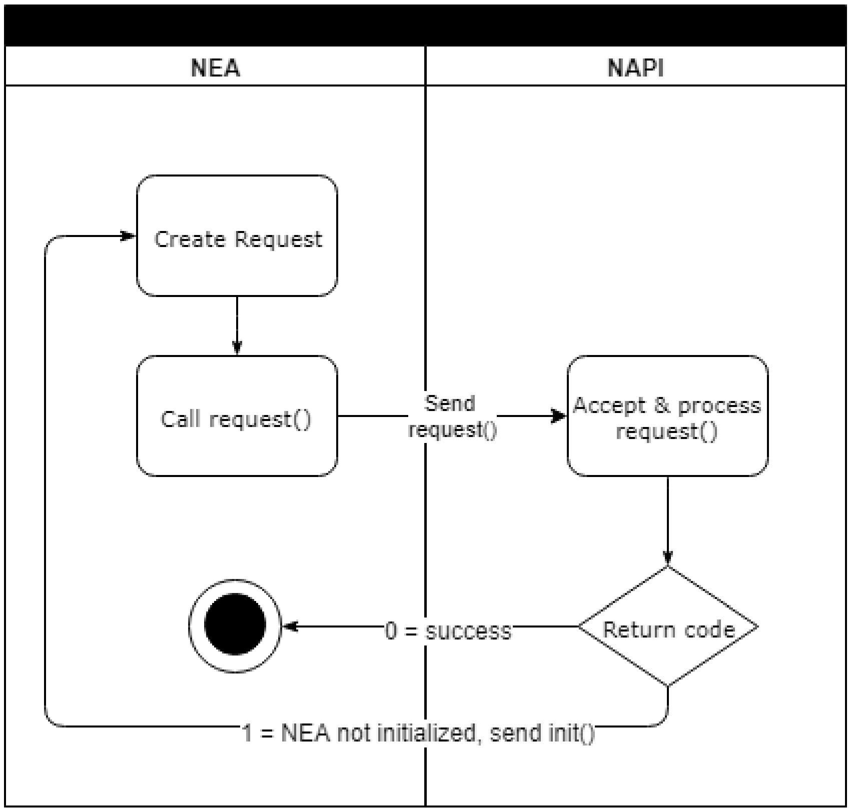

Figure 1: Request function workflow

1. Create the request in a memory buffer and pass the request to NAPI
2. `Nymi API` creates a copy of the request message.
3. `Nymi API` initiates the requested operation.

   The *request()* call returns 0 when Nymi API accepts the message and returns a 1 when NEA the has not been initialized. The NEA must run the *init* operation before `Nymi API` can accept any messages other than *init*.

The request message is a null-terminated string containing a JSON object with the following key-value pairs:

```
{
 "operation": "operation_name",
 "exchange": "exchange_string",
 "payload": {
   "property_name": "property_value",
   "property_name1": "property_value1"
   …
   "property_nameX": "property_valueX"
   }
 }
```

where:

- *operation_name* defines the operation for NAPI to perform. For example, *init*, *assert_identity*, *presence* and *lookup*.

subscribe operation

The `subscribe_endpoint` operation allows a Nymi-enabled Application (NEA) to change the Nymi Bluetooth Endpoint to which it is subscribed..

The subscribe_endpoint operation allows an NEA to change the `Nymi Bluetooth Endpoint` to which it is subscribed.

By default, each NEA is matched to it's local endpoint based on the IP address of the workstation. In most deployments, the NEA and endpoint are correctly matched by default, and connect automatically.

`subscribe_endpoint` request operations appear in the following format:

In central deployments, certain network configurations, such as workstations that have multiple network interfaces, may interfere with the automatic matching of the NEA and `Nymi Bluetooth Endpoint`. In these cases, the subscribe operation must be used by the NEA to communication to which workstation it wants to connect.

```
{
"operation": "subscribe_endpoint",
"exchange":"exchange_value",
"payload": {
"endpoint_id": "bar"
}
}
```

where:

- *operation* is the subscribe_endpoint.
- *exchange* is any value and is used to match the response to the request.

payload:

- *endpoint_id* is based on the endpoint IP address.

The subscribe_endpoint operation returns status codes only, no errors are returned. The following table displays possible status codes:

```
{
“Operation”: “subscribe_endpoint”,
"exchange":"exchange_value",
“payload”: {}
“status”: 0,
“error”: {}
}
```

An NEA can only be subscribed to one endpoint at any given time. When a subscribe operation is requested, the NEA is automatically unsubscribed from the endpoint it was previously subscribed to. If any Nymi Bands were present on that endpoint, they will become absent, and the NEA will receive corresponding presence update notifications. The NEA will then receive a Bluetooth status notification. If the requested Nymi Bluetooth Endpoint has connected successfully and is in a ready state, the NEA will receive a ble_ready notification, followed by presence update notifications for any present bands on that endpoint. Otherwise, the NEA will receive an error message. See *Bluetooth Notifications* for more information about possible error messages.

Note: The NEA will remain subscribed to the requested endpoint_id even if it is not able to connect to that Nymi Bluetooth Endpoint. If the Nymi Bluetooth Endpoint becomes ready at a later time (for example, that workstation is powered on), the NEA will receive a ble_ready message at that time.

init operation

The *init* operation initializes NAPI, configures communication channels between components, and performs certificate enrollment when required. Ensure that *init* is the first operation that is requested by the NEA. When the *init* operation succeeds, it is not necessary to call *init* again.

Initialization Options

There are three ways to call the *init* operation when initializing with certificate enrollment.

- nea_name
- nea_name + nes_url + token
- nea_name + nes_url + token + otp

JSON Object Format

Define the JSON payload for the *init* in the following format.

```
{
"operation": "init",
"exchange": "exchange_value",
"payload": {
"nea_name": "name_of_application",
"nes_url": "https_url_to_nes",
"token": "token",
"otp": "one_time_password",
"log_path": "path",
"url": "ws://agent_server:9120/socket/websocket",
}
}
```

where:

- *name_of_application* is the name that you assign to the NEA and is always required. The NES active group policy configuration influences the name that you can specify, in the following way:
    - When **`Manual OTP mode`** is enabled, you must specify the name as `NEAs`.
    - When **`Manual OTP mode`** is not enabled, you can assign any name to the NEA.

  Contact the NES Administrator to determine the active group policy configuration settings.
- *nes_url* field is the URL for the NES website application. You require this parameter in the first *init* call. The format of the URL is https_url_to_nes
- *token* is an HTTP Bearer token that NES uses to authenticate the NEA user or computer. This parameter is optional. If you will use this parameter, you must specify it in the first *init* call. Obtain the token as described in the *Appendix*.
- *one_time_password* is the OTP that provides the NEA with the ability to generate the NEA certificate. Include *one_time_password* in the payload when Manual OTP mode is configured in the active group policy in NES. You require this parameter in the first *init* call. When you define this parameter, you must also define the *https_url_to_nes* and *token* parameters.
- *path* is the log file path on the development machine. If you do not specify the path property, the NEA uses the default log path, which is your current working directory.
- *agent_server* specifies the hostname of the machine that runs the `Nymi Agent` service.

`Example`

The following code block provides an example of a JSON object that instructs NAPI to initialize the NEA that requires an OTP to retrieve a certificate.

```
{
"operation": "init",
"exchange": "rAndOm_IdeNtifyiNG_StrING_1211",
"payload": {
"nea_name": "NEAs",
"nes_url": "https://server-2.nymi.lab/nes",
"token": "eyJVc2VyVG9rZW5TdHJpbmciOiJMbk..",
"otp": "4C82F6CF3ABED723",
"url": "ws://agent.nymi.com:9120/socket/websocket"
}
}
```

Results

A successful *init* operation produces a response with the following properties.

```
{
"operation": "init",
"exchange": "rAndOm_IdeNtifyiNG_StrING_1211",
"payload": {}
"status": 0,
"error": {}
}
```

An unsuccessful *init* operation produces a non-zero status.

The following table summarizes the status codes that can appear, and the payload properties that you require for a subsequent *init* call.

Table 2: Init Status Codes

Status code	Payload properties for subsequent *init* call
0	Operation completed successfully with the defined payload. The system is initialized. Additional calls to *init* are not required.
11xx	Operation completed successfully with the defined payload. When a request other than *init* is sent before the system is initialized, the system returns a status code 1100. If the system was already initialized, but a request for *init* was sent, the system returns a status code 1110.
8000	Payload is missing the *token* and *nes_url* property definitions. Call *init* again and include the *token* and *nes_url* properties, in addition to the *nea_name*.
8100	Payload includes the *token* and *nes_url* but is missing the OTP property definition. Call *init* again and include the *otp* in the payload, in addition to the *nea_name*, *token* and *nes_url* properties.
9000	There was an issue with the certificate from NES. Contact the NES Administrator for assistance.

The following flowchart provides an overview of how you can use NAPI responses to an *init* call, to determine the properties that you need to include in the payload file.

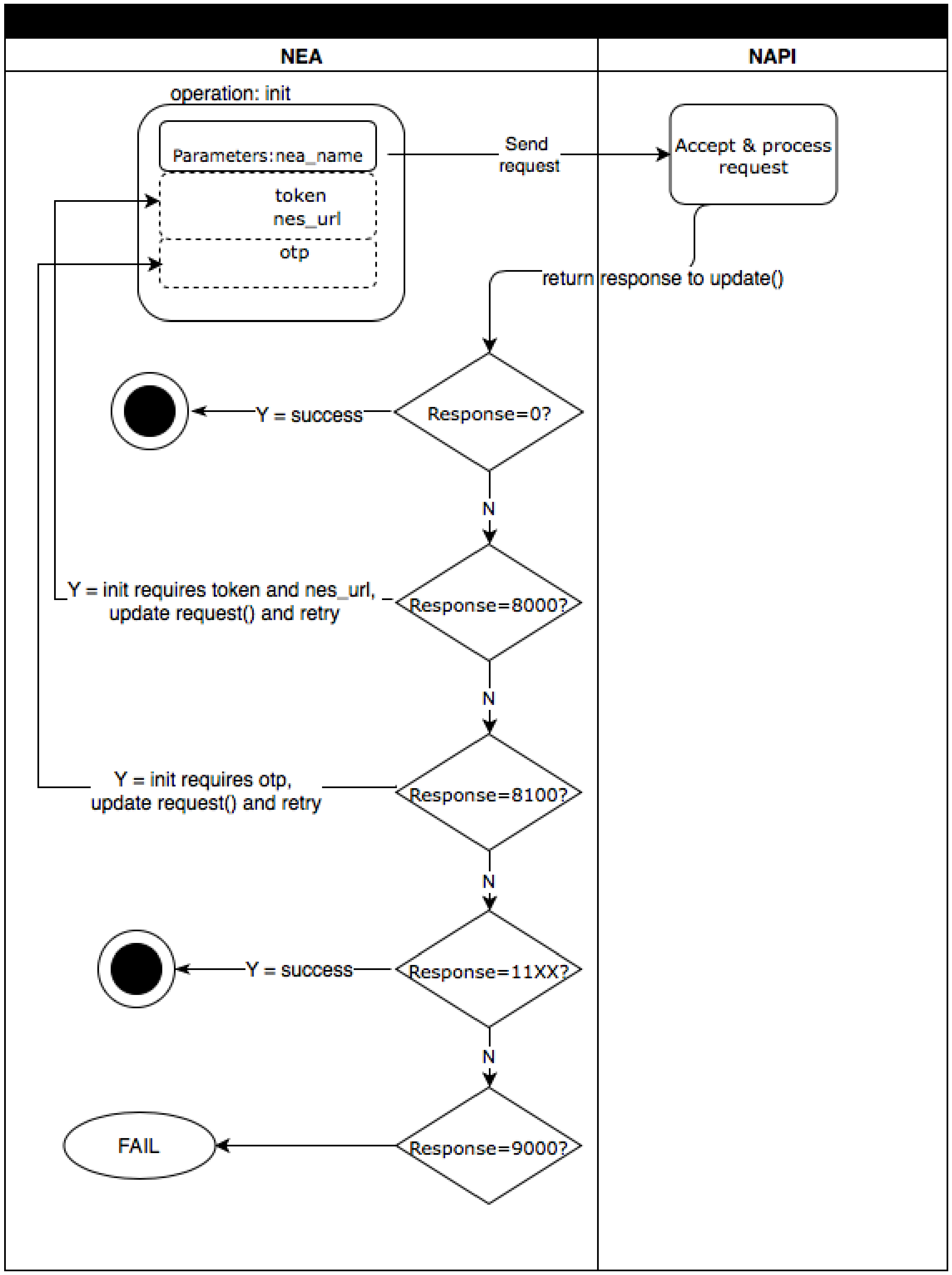

Figure 2: NAPI response calls to init

lookup

An NEA requires the device ID of a Nymi Band to communicate with the Nymi Band. You can retrieve the device ID of a Nymi Band from NES by using the *lookup* operation.

Use the *lookup* operation to determine the following values:

- Device ID

  Note: Device operations require that you specify the Nymi Band (or device) ID value that appears in the response.
- NfcUID of the Nymi Band
- Domain and name of the user.
- User status in Active Directory (AD). The AD status for a user appears in the response when user status check is enabled in NES. The following table summarizes the possible user statuses.

Table 3: AD user statuses

User Status	Definition
Active	User account is enabled.
NotExist	User account was deleted from AD.
Inactive	User account is disabled.
Locked	User account is locked. This status can appear with Active and Password Expired.
PasswordExpired	User account has an expired password. This status can appear with Active and Locked.

By default, NES disables support for user status checks in AD. Contact the NES administrator to enable AD user status checking, and optionally the checking interval in the `NES Administrator Console`.

JSON Object Format

Define the *payload* JSON object for the *lookup* command in the following format.

```
{
"operation": "lookup",
"exchange": "exchange_value",
"payload":
{
"nes_url": "https_url_to_nes",
"query": "query_JSON",
"lookup_keys": "key_JSON"
}
}
```

where:

- *nes_url* field is the URL for the NES website application. You require this parameter in the first *init* call. The format of the URL is https_url_to_nes
- *query_JSON* is a JSON object that defines the query values. Acceptable values include *NfcUID*, *Domain* and *Username*, and *NymiBandID*.
- *key_JSON* is a JSON array that contains a list of values that you want to appear in the response. Supported values include *NfcUID*, *Domain* and *Username*, *NymiBandID*, and *UserStatus*.

Note: The property names *Domain* and *Username* are case-sensitive.

Example 1

The following code block provides an example of a JSON object that instructs NAPI to provide the NfcUID of a device and the user status for a user named *JSmith* in the *MyCorpDomain* domain.

```
{
"operation": "lookup",
"exchange": "rAndOm_IdeNtifyiNG_StrING_1218",
"payload": {
"nes_url": "https://nes.nymi.com/nes/",
"query": {
"Domain":"MyCorpDomain",
"Username": "JSmith"
}
"lookup_keys": ["NfcUID", "UserStatus"]
}
}
```

Results 1

A successful *lookup* operation produces a response with the following properties.

In this example, the check user status in AD option is enabled in NES, as a result, the response includes the *UserStatus* property.

```
{
"operation": "lookup",
"exchange":"rAndOm_IdeNtifyiNG_StrING_1218",
"payload": {
"lookup_values":{"NfcUID": "1234xyz", "UserStatus":"Active|PasswordExpired"},
},
"status": "0",
"error: {}
}
```

Example 2

The following code block provides an example of a JSON object that instructs NAPI to provide the NfcUID of a device with Nymi Band (or device) ID *"C2:FA:D7:F0:D7:96"*.

```
{
"operation": "lookup",
"exchange": "rAndOm_IdeNtifyiNG_StrING_1218",
"payload": {
"nes_url": "https://nes.nymi.com/nes/",
"query": {
"NymiBandID": "C2:FA:D7:F0:D7:96"
}
"lookup_keys": ["NfcUID"]
}
}
```

Results 2

A successful *lookup* operation produces a response with the following properties.

```
{
"operation": "lookup",
"exchange":"rAndOm_IdeNtifyiNG_StrING_1218",
"payload": {
"lookup_values": {"NfcUID": "1234xyz"},
},
"status": "0",
"error: {}
}
```

assert_identity

The *assert_identity* operation provides an NEA with the ability to confirm that a Nymi Band that is assigned to a specific user is authenticated and within Bluetooth range.

Trusted Communication

Trusted communication between the Nymi-enabled Application and the Nymi Band occurs using the *assert_identity* operation. The *assert_identity* command completes a cryptographic handshake with the Nymi Bandand verifies user and Nymi Band identity.

TLS protocol ensures that NEAs and Nymi Bluetooth Endpoint connect to the trusted / Agent Nymi Security Protocol secures communication between and Nymi Bands.

The *assert_identity* command completes a cryptographic handshake with the Nymi Band and verifies user/band identity.

Note: The Nymi Band must be in an authenticated state when you call the *assert_identity* operation.

JSON Object Format

Define the *assert_identity* JSON object in the following format.

```
{
"operation": "assert_identity",
"exchange": "xchange_value",
"payload": {
"nes_url": "https_url_to_nes",
"device": "NymiBandID",
"assert_type": "assert_user"
}
}
```

where:

- *nes_url* field is the URL for the NES website application. You require this parameter in the first *init* call. The format of the URL is https_url_to_nes
- *NymiBandID* is the Nymi Band (or device) ID value that is returned in the *lookup* result.

Example

The following code block provides an example of a JSON object that instructs NAPI to assert the identity of the user with device ID *C2:FA:D7:F0:D7:96*.

```
{
"operation": "assert_identity",
"exchange": "rAndOm_IdeNtifyiNG_StrING_5555",
"payload": {
"nes_url": "http://nes.nymi.com/nes/",
"device": "C2:FA:D7:F0:D7:96",
"assert_type": " assert_user "
}
}
```

Results

A successful *assert_identity* operation produces a response with the following properties.

```
{
"operation": "assert_identity",
"exchange":"rAndOm_IdeNtifyiNG_StrING_5555",
"payload": {
"Username": "Jsmith",
"Domain": "Corp"
},
"status": "0",
"error: {}
}
```

presence update

Using the presence update request, you can retrieve the current state of the Nymi Band. Presence update requests are non transactional. The presence request has no response and a presence response is not tied to a specific request.

When `Nymi API` sends a presence update request, the system replays the last presence update that was received. When the presence state changes, a Nymi-enabled Application receives an automatic notification.

Presence is relative to an endpoint (the response indicates if the Nymi Band is in range of the NEA). A Nymi Band can be present on some endpoints, but absent on others. If the presence state is false the presence state returns as `absent`.

JSON Object Format

Define the *presence* request JSON object in the following format.

```
{
"operation": "presence",
"exchange":"exchange_value",
"payload": {
"device": device
}
}
```

For presence notifications, refer to presence notifications on page 31.

Table 4: Presence Payload

Properties	Value	Description
Device	device	The Nymi Band MAC address.
State		The value named `state` has a string value.
	absent	The state is not detected. A Nymi Band that has not been reported on* should be considered the same as a Nymi Band that has most recently been reported `absent`. A Nymi Band that has an `absent` state may be unheard from for a certain length of time. **Note:** * reported on refers to a) The Nymi Band is connected via BLE and is present. b) It has sent a BLE advertisement to the endpoint within the last 30 seconds.

Properties	Value	Description
	unauthenticated	Nymi Band is not authenticated (may or may not have authenticators enrolled). A Nymi Band that is not authenticated may be on-body and `unauthenticated` or is being charged.
	weak	Nymi Band is in an authenticated state. The advertisement authentication code is not verified.

See *presence notification* for information about returned parameters.

Response Messages and Notifications

By default, a response message contains the `operation` value, the payload, and the `status` value of the request.

- Responses are messages that are generated as a result of the operations previously submitted to NAPI.
- Notifications are system-generated messages that provide information about state changes in the environment. Notifications are not generated in response to a request made by a function call.

- When the Presence of a Nymi Band changes, for example, when the `Nymi Agent` authenticates a Nymi Band.
- When a `Nymi Runtime` error occurs.

The *update* function retrieves the notifications and responses from memory. Before the response appears in the update queue, the system requires time to process the request and generate the response. Call the *update* function on a single thread, to maintain one centralized place that handles all *update* responses.

IMPORTANT: In large environments, call update frequently to avoid the loss of responses and notifications.

Response Messages and Notifications

- Responses are messages that are generated as a result of the operations previously submitted to NAPI.
- Notifications are system-generated messages that provide information about state changes in the environment. Notifications are not generated in response to a request made by a function call.

- When the Presence of a Nymi Band changes, for example, when the `Nymi Agent` authenticates a Nymi Band.
- When a `Nymi Runtime` error occurs.

The *update* function retrieves the notifications and responses from memory. Before the response appears in the update queue, the system requires time to process the request and generate the response. Call the *update* function on a single thread, to maintain one centralized place that handles all *update* responses.

IMPORTANT: In large environments, call update frequently to avoid the loss of responses and notifications.

Exchange Message

NAPI sends response messages and notifications to a memory buffer. There is only one response queue, and requests are not tracked against their original threads.

Define an exchange value in the *request_obj* to match the requests that are sent from various threads to the responses that are received on the *update* thread.

A response message appears in the following format:

```
{
"operation":"operation_value",
"payload": {
"property_name": "property_value",
"property_name1": "property_value1",
…
"property_nameX": "property_valueX"
}
"status": 0 or error_code,
"error": {
"error_description": "error_description",
"error_specifics": "specific error description"
}
}
```

Consider the following:

- *operation* always appears in the response and the value depends on the reason for the response.
  - For a request response, the *operation_value* matches the *operation_value* in the request.
  - For a notification response that is the result of an error, the *operation_value* is *error*.
- *payload* always appears in the response. If the *payload* does not contain properties or the response results in an error, the *payload* will appear empty. For example, `"payload": {}`.
- *status* is 0 when the operation is successful and an integer value that is greater than zero when the operation fails.
- *error* always appears in the response and the value depends on the reason for the response.
  - If the response is the result of a successful request, error is empty. For example, `"error": {}`.
  - If the response is the result of a failed request or error notification, status displays an error code, and error contains descriptive information about the failure. See *Error Handling* for more information.

update function

Use the *update* function to retrieve responses for requests and system notifications from NAPI.

The declaration for the update function is as follows:

```
typedef const char* (WINAPI* UPDATE_FUNC_POINTER)(int timeout_ms);
UPDATE_FUNC_POINTER update = NULL;
```

Where *timeout_ms* is an integer value that represents the number of milliseconds (ms) that the update function waits for a response before timing out.

Ensure that you do not call *update* simultaneously on two threads.

Results

The *update* function returns a pointer to a JSON message as an UTF-8 string. The string has one of the following values:

- Empty string, when a timeout occurs
- Valid JSON string

assert_identity response

The assert_identity request returns *Username* and *Domain.* properties

assert_identity Results

The *UserStatus* property is an optional property. The UserStatus is stored in the Active Directory (AD).

If the UserStatus option is set in the NES console in the *Policies* > *Active Directory* page, the Active Directory status appears in the assert_identity response. If the option is not set, it does not return in the response.

The *UserStatus* option has the following possible values:

User Status	Definition
Active	User account is enabled.
NotExist	User account was deleted from AD.
Inactive	User account is disabled.
Locked	User account is locked. This status can appear with Active and Password Expired.
PasswordExpired	User account has an expired password. This status can appear with Active and Locked.

The last three properties can be combined into a coma separated list.

By default, NES disables support for user status checks in AD. Contact the NES Administrator to enable AD user status checking, and optionally the checking interval in the `NES Administrator Console`.

A successful *assert_identity* operation produces a response with the following properties.

```
{
"operation": "assert_identity",
"exchange":"rAndOm_IdeNtifyiNG_StrING_5555",
"payload": {
"Username": "Jsmith",
"Domain": "Corp"
"UserStatus": "Active"

},
"status": "0",
            "error: {}
            }
```

initialization error notifications

After initialization, NAPI might disconnect from the `Nymi Agent`, which results in update retrieving an error notification similar to the following example.

```
{
  "operation": "error",
  "exchange": null,
  "payload": {},
  "status": 4000,
  "error": {
      "error_description": "Nymi Agent missing.",
      "error_specifics":""
  }
}
```

When a disconnect occurs, NAPI automatically attempts to reconnect to `Nymi Agent`. Any requests that an NEA performs will fail until it retrieves a `reconnection` notification.

A reconnection notification appears similar to the following:

```
{
"operation": "reconnection",
"exchange": "null",
"payload": {},
"status": 0,
"error": {}
}
```

presence notifications

When `Nymi API` (NAPI) detects a change in Nymi Band presence, NAPI generates a presence notification.

The *update* calls that you perform after you perform the *init* operation retrieve a sequence of presence notifications, one for each present Nymi Band (if any Nymi Bands are present within range. Presence updates are non transactional. The system will return any changes to presence.

It is recommended that you develop a method for your application that tracks when the Nymi Bands come in and out of range.

Presence notifications appear in the following format:

```
{
"operation":"presence",
"exchange":null,
"status":0,"
payload":{
        "device": device,
        "proximity":"proximity value",
        "service_request_state":service request state
        "state":state
},
"error":{}
}
```

where:

- *proximity_value*: determined by the distance between the Nymi Band and the BLE adapter. The *proximity_value* will change when the Nymi Band moves closer or farther from the BLE adapter. The threshold (distance) for the *proximity_value* is determined in the *nbe.toml* file.

  Note: To edit the *nbe.toml* file, refer to Editing the nbe.toml File on page 35.
- *state*: determined by the state of the Nymi Band; weak, absent, or unauthenticated.
- *service request state*: a flag that accompanies each presence notification and determines if there is a message in the Nymi Band that is ready to be downloaded. If the value of `service_request_state` is '1', the Nymi Band has a message. If the value is '0', there are no messages.

Note: If the payload contains only the device, no response is returned for this operation. A notification is returned, which is not tied to any request and does not contain any values.

Table 5: Proximity values for presence notifications

Proximity values	Definition	Example: Nymi Lock Control Behavior
4	The BLE adapter does not detect the Nymi Band.	For example, the user may be in another room. When the user enters the BLE adapter range, the *proximity_value* will go from 4 to 3. Nymi Lock Control does not perform any actions. When the user leaves the BLE adapter range, the *proximity_value* goes from 3 to 4. Nymi Lock Control does not perform any actions.
3	The BLE adapter detects the presence of the Nymi Band.	For example, the user is in the same room as their user terminal. When the user moves closer to the BLE adapter, the *proximity_value* will go from 3 to 2. Nymi Lock Control does not perform any actions. When the user moves further from the BLE adapter, the *proximity_value* goes from 2 to 3. Nymi Lock Control locks the user terminal if it is unlocked.

Proximity values	Definition	Example: Nymi Lock Control Behavior
2	The BLE adapter is close to the Nymi Band.	For example, the user is near their user terminal. Nymi Lock Control keeps the user terminal unlocked while the user remains within this range (*proximity_value* is 2 or less). While Nymi Lock Control is enabled, the user may press the Enter key or the space bar on their keyboard to unlock their user terminal. When the user moves the Nymi Band closer to the BLE adapter, the *proximity_value* goes from 2 to 1. Nymi Lock Control will allow the user to access their user terminal without entering their credentials. When the user moves the Nymi Band further from the BLE adapter, the *proximity_value* goes from 1 to 2.
1	The BLE adapter and the Nymi Band are in very close range.	For example, the user may be sitting at their user terminal. When the user moves the Nymi Band closer to the BLE adapter, the *proximity_value* goes from 1 to 0. This initiates a tap intent. When the user moves the Nymi Band away from the BLE adapter, the *proximity_value* goes from 0 to 1. This ends a tap intent.
0	The BLE adapter and the Nymi Band are adjacent (within 4 inches or 10 cm).	For example, the user places their Nymi Band on top of their BLE adapter. A tap intent is in progress and indicates a task.

Table 6: State values for presence notifications

State Value	Definition
Absent	The `Nymi Agent` cannot communicate with the Nymi Band. This state also applies when a user wears an unenrolled Nymi Band. Reasons for Nymi Band absence include: • Nymi Band has been removed from the body. • Nymi Band has not communicated with the `Nymi Agent` for at least 30 seconds. • Nymi Band has not been within the range of the BLE Adapter for at least 30 seconds.
Unauthenticated	Nymi Band is enrolled and but not authenticated.
Weak	Nymi Band is in an authenticated state.

Bluetooth notifications

Nymi Bluetooth Endpoint is a client service that communicates with the Bluetooth Adapter. Bluetooth notifications for Bluetooth Adapter status are non-transactional.

The Bluetooth Adapter communicates to the Nymi Band. Each time that a Bluetooth Adapter becomes available, the *update* function retrieves a notification in the following format.

```
{
  "operation": "ble_ready",
  "exchange": null,
  "status": 0,
  "payload": {},
  "error ": {}
}
```

If a Bluetooth Adapter becomes unavailable, the *update* function retrieves an error notification in the following format.

```
{
  "operation": "error",
  "exchange": null,
  "payload": {},
  "status": "error_code",
  "error": {
      "error_description":"error_description>",
      "error_specifics":"error_specifics"
   }
 }
```

where *error_code* is one of the following values: 5000, 5010, 5100.

For more information about error codes, see *Error Handling*.

Intent Notification

An intent occurs when a user taps their authenticated Nymi Band next to an NFC reader or Bluetooth radio antenna, and is used to signal an intent to take an action. For example, an intent to provide an e-signature is generated when a user taps their authorized Nymi Band against an NFC reader.

A NES server must be specified in the `init` message in order for intent notifications to be received.

Intent notifications appear in the following format:

```
{
"operation": "intent",
"exchange": null,
"payload": {
"device": "MAC address",
        "type": "see below",
        },
        "status": 0,
        "error": {}
        }
```

where *device* is the Nymi Band device ID.

type is used to identify the manner in which the action was initiated.

type field	description
ble	A user tapped an authenticated Nymi Band against a BLE device or is in close proximity to a BLE radio antenna, such as a BLE adapter or built-in Bluetooth receiver.
nfc	A user tapped an authenticated Nymi Band against an NFC reader or is in close proximity to read range of the NFC reader.

Status Codes

A 2201 status code is reported when the NFC reader is unsuccessful at mapping the NFC ID to the enrolled Nymi Band.

A 2200 status code is reported when a NES communication error (for example, NES is offline) occurs.

Note: The 2201 and 2200 status codes do not contains a device ID in the payload.

Editing the nbe.toml File

A backup configuration file is installed on the user terminal when the `Nymi Bluetooth Endpoint` is installed or updated. This file, *nbe.default.toml*, contains the default values that control BLE tap behavior with the Nymi Band and BLE adapter. Use the values in the *nbe.default.toml* file as a template for the *nbe.toml* file. These files are located in *C:\Nymi\Bluetooth_Endpoint\* .

Note: `Nymi Bluetooth Endpoint` will only recognize RSSI values in the *nbe.toml* file. Retain a backup of a useful configuration by copying the *nbe.toml* file and renaming it.

Table 7: Default configuration settings for `Nymi Lock Control` and BLE tap intent

nbe.toml **Entry**	**Default Value**	**Description**
agent_url	"ws://127.0.0.1:9120/ socket/websocket" (do not change)	Identifies the location of the agent URL. The default value shown in this table is generated if the agent is installed locally. If the agent URL is installed centrally (via remote installation), the hostname of the URL will be different. **The agent_url must be present when using an *nbe.toml* file.**
rssi_window_tap	10	This determines the duration the Nymi Band must be within tap-distance of the BLE radio antenna to complete a tap. A larger value increases the duration required to perform and decrease the sensitivity.
rssi_window_long	50	This determines the frequency that `Nymi Bluetooth Endpoint` checks the distance between the BLE radio antenna and the Nymi Band. `Nymi Bluetooth Endpoint` tracks trends in these changes to trigger a Nymi Lock Control action, such as **`keep unlocked when present`**, **`lock when away`**, or **`unlock when present`**.
rssi_tap_threshold	0 (must be 0 or negative)	This determines the range at which a tap event will occur. A smaller negative value means a closer distance to the BLE antenna. BLE tap is disabled by default (value = 0). **Enter a non-zero, negative number to enable BLE tap**. Nymi recommends an RSSI value of -42. If the Nymi Band maintains a minimum distance specified by *rssi_tap_threshold*, for a duration *rssi_window_tap*, a BLE tap is performed.

nbe.toml Entry	Default Value	Description
rssi_cutoff_close	-70 (must be 0 or negative)	This determines the outer range of the close distance-threshold (excluding tap distance) for Nymi Lock Control. Enter 0 to bypass the proximity functionality of Nymi Lock Control. If the Nymi Band maintains a close distance to the BLE radio antenna and the RSSI values measured are within the *rssi_cutoff_close* value, Nymi Lock Control keeps the user terminal unlocked. If the Nymi Band moves away from the BLE radio antenna, and the RSSI values measured are on a decreasing trend and goes from the *rssi_cutoff_close* value to the *rssi_cutoff_far* value, Nymi Lock Control locks the user terminal.
rssi_cutoff_far	-75 (must be negative)	This determines the outer range of the far distance-threshold (excluding tap distance) for Nymi Lock Control. If the Nymi Band moves towards the BLE radio antenna, and the RSSI values measured are on an increasing trend and goes from the *rssi_cutoff_far* value to the *rssi_cutoff_close* value, Nymi Lock Control unlocks the user terminal.

1. Make a copy of the *C:\Nymi\Bluetooth_Endpoint\nbe.default.toml* file, and name the file *nbe.toml*.
2. Edit the *nbe.toml* file with a text editor.
3. Edit the RSSI values in the file. Refer to the descriptions in the table above.
4. Save the *nbe.toml* file.
5. Restart the `Nymi Bluetooth Endpoint`.
   a) Press the Windows key on the keyboard, or click the start button on the toolbar. Enter "Services" in the search bar. The Services application window appears.
   b) Search for **`Nymi Bluetooth Endpoint`** in the Services application.
   c) Right-click **`Nymi Bluetooth Endpoint`** and restart it.

Once restarted, the `Nymi Bluetooth Endpoint` application will be updated with the edits made in the *nbe.toml* file. Updated BLE tap intent and `Nymi Lock Control` settings will be implemented on the user terminal. If the *nbe.toml* file is not present, `Nymi Bluetooth Endpoint` behaves under default settings.

Error Handling

The *update* function retrieves errors in the following scenarios.

- When a *request* operation fails, the response contains a non-zero "`status`" and *error* contains information about the failure. For example, when the *assert_identity* request was called with an incorrect *nes_url* value.
- When an *update* receives a notification response from NAPI as the result of a runtime error, the operation value is "`error`". For example, when the BLE adapter is removed from the USB port.

Notifications and response messages that result in an error appear in the following format:

```
{
  "operation": "operation_value",
  "exchange": "null" or "exchange_value",
  "payload": {}
    "status": status_code,
    "error": {
    "error_description": "general error description",
    "error_specifics": "specific error description"
  }
}
```

where:

- *operation_value* provides the operation value for the response or notification. For a response, the value is the same value that appeared with the request. For a notification, the value is `error`.
- *payload* does not contain any properties.
- *exchange* contains the user-defined exchange value, as it appeared in the request. If an exchange value was not specified in the request, the exchange value is `null`.
- *status_code* provides the status code that is associated with the error. See the *Status codes* table for more information
- *error_description* provides the description of the error that is associated with the status code.
- *error_specifics* provides additional information about the source of the error. For example, when a request specifies invalid parameters.

The following table summarizes the values that can appear in the *status_code* and *error_description*.

Status Code

Nymi provides you with status codes that assist you in solving SDK code-related issues and errors.

Table 8: Status codes

Status code	Error description
0	Operation completed successfully.
1000	Request made with invalid JSON.
1100	Request other than *init* sent before initialization.
1110	*init* request sent when already initialized.
1200	Cannot connect to NES. NES URL not specified in *init*.
2000	Request made with invalid parameters.
2102	Request made with device that does not exist. This is a permanent error, retries will fail.
2200	Problem occurred while communicating with NES.
2201	The requested query was not found on NES.
3000	Operation timed out. Retry the operation.
3010	Operation interrupted. For example, when the battery dies.
3100	Operation made during invalid band state.
4000	Connection to `Nymi Agent` lost. When you see this error, requests fail until *update* retrieves a reconnection notification.
4010	Request made while disconnected from `Nymi Agent`.
5000	Something went wrong with the Bluetooth Adapter.
5010	The Bluegiga BLED112 dongle is missing.
5100	`Nymi Bluetooth Endpoint` is missing or stopped.
6000	A temporary, recoverable error that indicates that the Nymi Band is currently not able to perform the operation, but the operation might succeed if the NEA tries the operation again.
7000	Error originating from the Nymi Band. Applies to device operations only.
8000	*init* payload requires the *token* and *nes_url* properties.
8001	NEA data is corrupt or not accessible.
8002	Missing organization name in the L1 certificate.
8100	*init* payload requires *otp* property.

Status code	Error description
9000	An error occurred. See *error_specifics* for more details.

Note: Status codes 1000 and 2000, should be considered the same as they indicate a messaging issue (for example, invalid JSON).

Troubleshooting

NAPI writes information to log files that allow you to monitor and troubleshoot the NEA.

For additional assistance, visit the Support page on the Nymi website, or contact your Nymi Solution Consultant.

The following table summarizes the log files that are available for troubleshooting.

Table 9: Log file locations

Component	Log location
NAPI	By default, the current working directory.
`Nymi Agent`	*/usr/lib/nymiagentd/nymi_agent.log*
`Nymi Bluetooth Endpoint`	*/var/log/nymi/nymi_bluetooth_endpoint.log*

Enable debugging

Debug logging is available, but should not be enabled in a production environment.

On CentOS 7, you can enable the debug logging for the Nymi Agent, and Nymi Bluetooth Endpoint by editing each of their service files.

1. On CentOS 7, navigate to the service files located at:

   `nbed - Nymi Bluetooth Endpoint Daemon`

   - service file location/directory: /etc/systemd/system/nbed.service

   `nymiagentd - Nymi Agent Daemon`

   - service file location/directory: /etc/systemd/system/nymiagentd.service
2. Add the following text to the [Service] section of each of the files:*Environment=NYMI_DEBUG=1*
3. Next, run the *$ sudo systemctl daemon-reload* command.
4. Next, run *$ sudo systemctl restart nbed* command.
5. Next, run *$ sudo systemctl restart nymiagentd* command.
6. Restart the Nymi Bluetooth Endpoint services.

Service information and runtime errors

You can run the following command to view information about the installation and runtime errors.

```
$ journalctl -xe
```

Checking the service status

Check the status of the nbed service by running the following command:

```
$ systemctl status nbed
```

Check the status of the nymiagentd service by running the following command:

```
$ systemctl status nymiagentd
```

Starting services commands

By default, the services start automatically, but you can run the command whenever you need to start the services.

Optionally, start the services using the following commands:

```
$ sudo systemctl start nbed
```

```
$ sudo systemctl start nymiagentd
```

The `NymiAgentd` is installed in the following directory:*/usr/lib/nymiagentd*. The NBEd binary is located in the following path:*/usr/sbin/nbed*.

Appendix

Review this section for supplementary information about `Nymi API C Interface`.

Authentication requirements

An NEA and the Nymi Band establish trusted communication by using certificates. The first time that a user runs the NEA, the NEA retrieves a certificate from NES. The NEA certificate is stored in a keystore. Access to the keystore, by default, is enabled for all users

The NES Administrator can configure automatic or manual certificate retrieval. When the NES Administrator configures manual certificate retrieval, to initiate the retrieval process, you must specify a one-time password (OTP) in the *init* operation.

Acquire an Authentication Token

The first operation that the NEA must call is an *init* operation. If the *init* call results in a status code 8000, the (NEA must make an HTTP request to the NES REST API and acquire a token).

You can access NES by using one of the following endpoints to acquire the initial token:

- Basic Authentication
- Basic Authentication with cookies
- Negotiate

Basic Authentication (https://AS_url/api/BasicLoginWithToken)

This endpoint requires you to pass the user credentials in the authorization header.

A successful call:

- Returns one of the following outputs:
  - When Accept Header is set to application/xml or application/shtml+xml, the following xml output:

```
 <LoginWithTokenResult xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
schemas.datacontract.org/2004/07/Providers.Interfaces">
 <Success>true</Success>
 <Token>
 …
 </Token>
 </LoginWithTokenResult>
```

  - When the Accept header is not defined, the following JSON string:

```
 {"Success"="true", "Token"="<token>"}
```

- Passes the token in the WwwAuthenticate header.

Basic Authentication with Cookies (https://AS_url/api/BasicLoginWithCookies)

This endpoint requires you to pass the user credentials in the authorization header.

A successful call:

- Returns the following JSON string:

  ```
  {"Success"="true", Cookies={"cookie1": "value1", "cookie2": "value2"}}
  ```

- Pushes the token as a *NymiAuth* cookie.

Published in Canada.
Nymi Inc.
Toronto, Ontario
www.nymi.com