Nymi API WebSocket Interface Guide

Nymi Enterprise Edition
v2.0
2020-09-18

Contents

Preface

Nymi™ provides periodic revisions to Nymi Enterprise Edition™. 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 `Nymi Enterprise Edition` documentation suite.

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
01	December 20, 2019	This guide is reissued due to document version update. There are no content changes since NEE 2.6.0.
02	September 18, 2020	Reissued to clarify behaviour for the presence operation, presence notification and bluetooth notifications.

Related documentation

- **Nymi Enterprise Server Deployment Guide**

  This document provides an overview of the components and the steps that are required to deploy the NES.
- **Nymi Enterprise Edition Administration Guide**

  This document provides information about how to use the `NES Administrator Console` to manage a `Nymi Enterprise Edition` system. This document describes how to set up, use and manage the Nymi Band™, and how to use the `Nymi Band Application`.
- **Nymi Enterprise Edition Release Notes**

  This document provides supplemental information about Nymi Enterprise Edition, including new features, limitations, and known issues with the Nymi Band, NES, the Nymi Band Application, and the NES Administrator Console.

- **Nymi Enterprise Edition 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 WebSocket Interface Overview

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

`Nymi API WebSocket Interface` supports enterprise integration and deployment of features that are available in the Nymi Enterprise Edition software. This guide provides information about how to develop a Nymi-enabled Application using the `Nymi API WebSocket Interface`.

Nymi SDK Overview

Nymi offers an SDK that delivers an API over an RFC-6455 compliant WebSocket and accessed using standard web clients that support WebSockets.

SDK Package Contents

The Nymi SDK package contains the following artifacts:

- Web Sample (JavaScript)
- BleDriver_xx.msi
- Nymi Runtime Installer

The Nymi SDK contains components that enable you to build Nymi-enabled Applications (NEA). The `Nymi Runtime` consists of the `Nymi Agent` and the `Nymi Bluetooth Endpoint`

`Nymi Bluetooth Endpoint` is a service deployed on individual workstations to provide local BLE communication with Nymi Bands through the Nymi-provided Bluegiga Adapter. The `Nymi Agent` facilitates communication between NEAs and the Nymi Band, and maintains knowledge of the Nymi Band presence and authenticated states.

The Nymi Agent acts like a WebSocket and contains the Nymi Security Library. The `Nymi Agent` acts as a server to which the `Nymi Bluetooth Endpoint` and Nymi-enabled Application connect and passes data between the two. Communication between each of these components is also carried on WebSockets, although these sockets are distinct from the `Nymi API WebSocket Interface` and run on a different TCP port.

`Nymi API WebSocket Interface` is technically a sub-component of the `Nymi Agent` and is installed with it, but acts independently from the `Nymi Agent`; and is entirely optional. This component is the focus of this guide.

Nymi API WebSocket Architecture

This guide contains information that helps you to understand and develop Nymi-enabled Applications. Nymi recommends that you read the guide before beginning development.

The Nymi API WebSocket Interface provides Nymi developers with an alternative way to utilize the functionality of the Nymi SDK, over a WebSocket connection managed by a web-based or other application. The Nymi API WebSocket Interface allows developers to write web applications that access services available from the Nymi Enterprise Edition solution.

Note: The Nymi API WebSocket Interface is supported on Microsoft Windows platform only.

Nymi API WebSocket Interface

The Nymi API WebSocket Interface provides bi-directional communication using requests/responses over a persistent connection. All messages sent and received are encoded in JSON format. The architecture provides continuous communication using WebSocket connections between the Nymi Agent and Nymi-enabled Application (NEA) running either as a native application or inside of a web client.

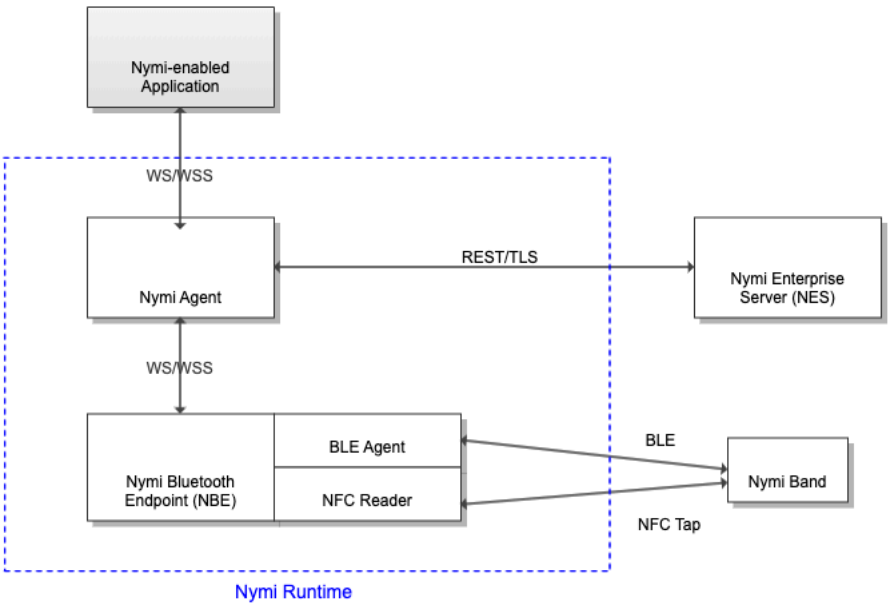

Figure 1: Nymi API WebSocket Interface Communication

The Nymi API WebSocket Interface communicates with Nymi Bands over a WebSocket client and supports multiple NFC readers. To enable NFC support, the NFC reader must be connected to a terminal with a compatible version of the Nymi Bluetooth Endpoint installed. Applications utilizing this functionality allow the Nymi Band to tap an NFC device, which sends an intent event message to the Nymi Agent.

Accessing the Nymi API WebSocket Interface, should be performed using Transport Layer Security (TLS) protocol. This requires a server certificate that is issued by a public or enterprise Certificate Authority (CA), which is trusted by clients.

Configuration parameters are set using a toml file. For more information see the `Configuration Overview` section.

WebSocket Keepalive Message

Nymi implements keepalive messages according to the RFC-6455 WebSocket Protocol standard for bi-directional communication. Nymi sends a ping message every 30 seconds to the Nymi-enabled Application and expects to receive a pong message response. The keepalive message indicates that the connection is still responsive.

Nymi supported web browsers, see the *Supported Platforms and Web Browsers* page, must send a pong message, which is sent in response to the ping control frame message. The pong control frame message ensures that the session is connected to the `Nymi Bluetooth Endpoint`. Nymi supported web browsers do not require any additional configuration to support this functionality.

If you are using a custom WebSocket client in Python or C# programming languages, additional implementation may be required.

Note: The WebSocket client, which is the Nymi-enabled Application, disconnects from the `Nymi Agent` if there are no messages (including pings and pongs) sent or received for a time period of 60 seconds.

Certificates Keystore

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. When using `Nymi API WebSocket Interface`, the keystore is stored on the machine running the Nymi Agent.

For supported Microsoft Windows platforms (Windows 10, Windows 7 64-bit and Windows 7 32-bit), the keystore is located at:

- *- %APPDATA%\Roaming\Nymi*

Alternative locations are:

- *C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\Nymi is for Local Service*
- *C:\WINDOWS\system32\config\systemprofile\AppData\Roaming\Nymi for LOCAL SYSTEM (64-bit binary)*
- *C:\WINDOWS\SysWOW64\config\systemprofile\AppData\Roaming\Nymi for LOCAL SYSTEM (32-bit binary)*

Supported Platforms and Web Browsers

Platforms

The Nymi Agent is implemented on the following platforms.

- Microsoft Windows 10, 64-bit
- Microsoft Windows 7, 32-bit and 64-bit

Web Browsers

The `Nymi API WebSocket Interface` works with most modern web browsers. It has been tested with:

- Firefox 70 or later
- Chrome 78 or later
- Internet Explorer 11 or later
- Microsoft Edge 44.18362.387.0

Note: If you have issued your own TLS root certificate using a private certificate authority (CA), you need to import the certificate into Firefox web browser. See *https://wiki.mozilla.org/CA/AddRootToFirefox* in the Mozilla documentation for more information.

Note: When using version 70 or later or of the Firefox web browsers, see the *Importing TLS Certificate into Firefox* for important information about importing a TLS certificate.

Importing the TLS Certificate into Firefox

Before Firefox can open a WebSocket connection for the NEA, you need to import the TLS certificate.

1. Open Firefox web browser.
2. In the right pane, navigate to Options.
3. Select **`Privacy and Security`**.
4. Under **`Certificates`** click **`View Certificates`** and then select **`Authorities`**.
5. Click **`Import`** and select the TLS root certificate from your machine.
6. Click **`OK`**.
7. Run the Nymi API WebSocket Interface and open the WebSocket connection using Firefox.

Supported NFC Readers

The `Nymi API WebSocket Interface` supports multiple NFC readers. A list of supported NFC Readers is found in the *Hardware requirements* section of the Nymi Enterprise Edition Administration Guide 2.6.0.

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 WebSocket Interface Sample Application

Nymi offers you a sample application that demonstrates some of the key functionality of the Nymi solution. The sample is a simple Javascript application that demonstrates all of the basic functions

supported by the API and allows a user to see both JSON request and response examples to help understand how the API works.

The sample applications are located within the package at: */nymi-sdk/windows/sampleApps*.

Deployment

The `Nymi API WebSocket Interface` can be deployed in a physical or virtual environment.

Deploying the Nymi API WebSocket Interface

The Nymi API WebSocket Interface is an added component installed during the installation of the `Nymi Agent` (the component to which Nymi-enabled Applications connect). Installation of the `Nymi API WebSocket Interface` is identical to the `Nymi Agent`. It can be deployed on each workstation and connected to a single local `Nymi Bluetooth Endpoint` and Nymi-enabled Application pair or can be deployed in a central location and used by multiple workstations. Because of the security implications of running a network server component on an uncontrolled workstation, Nymi recommends deploying the Nymi Agent and `Nymi API WebSocket Interface` on a central server, possibly NES.

The `Nymi Agent` can be deployed in both configurations simultaneously for different applications and, as such, existing NEE deployments can be extended with a central `Nymi Agent` and `Nymi API WebSocket Interface` without changing any other components or Nymi-enabled Applications.

Note: For more information, see the *Nymi Enterprise Edition Deployment Supplement Guide* for details on how to deploy in this configuration.

The `Nymi API WebSocket Interface` can be configured and deployed in environments such as Citrix and RDP. In this type of environment, the remote client is used to access a web browser that is used to connect to a Nymi-enabled Application.

The `Nymi Bluetooth Endpoint` and Nymi-enabled Application must know the identity of the workstation to which the application wants to connect. By default, this identity is the IP address of the workstation. For most deployments, both components use the same IP address, so they will connect automatically. In some deployments the IP address of the Nymi-enabled Application (the client IP) is different from the IP of the workstation. This will be the case, for example, in deployments where the Nymi-enabled Application is not running the same workstation as the Nymi Bluetooth Endpoint, such as remote desktop deployments. It may also be the case with certain network configurations. In these cases, the `subscribe` command must be used by the NEA to communicate to which workstation it wants to connect.

Note: See the `Nymi API WebSocket Interface` *Request, Operations and Response* section for details about the subscribe operation.

Installing Silently

You can install the `Nymi Bluetooth Endpoint` and the `Nymi Agent` using the commandline.

Use the following command to install the `Nymi Bluetooth Endpoint` only.

```
".\Nymi Runtime Installer 5.4.0.12.exe" -q InstallAgent=0
```

Use the following command to install the `Nymi Agent` only.

```
".\Nymi Runtime Installer 5.4.0.12.exe" -q InstallEndpoint=0
```

Managing Security and Certificates

System Administrators need to obtain and import certificates for your environment and for the `Nymi API WebSocket Interface`.

Certificates for the Nymi API WebSocket Interface

To ensure that trusted communication is configured, and TLS is used by the WebSocket, there are several certificates that you need to obtain and install:

- TLS Server Certificate chain including any intermediate CA certificates in base64 PEM format (note that the host certificate cannot be a wildcard certificate)
- Private key corresponding to the TLS server certificate, in base64 PEM format (the private key cannot be encrypted)
- Certificate of the root Certificate Authority (CA) issuing the TLS server certificate, in base64 PEM format

These three PEM files need to be installed at the location specified in the nymi_agent.toml file so that the `Nymi Agent` can communicate with the Nymi-enabled Application.

The `Nymi API WebSocket Interface` needs to connect to NES over HTTPS. The NES TLS server certificate must be issued by a Root CA trusted by the `Nymi API WebSocket Interface`. If the Root CA is not trusted by the workstation is should be installed in the Trusted Root Certification Authorities container for the local machine. See Microsoft documentation for information about installing Trust Root Certificates: https://docs.microsoft.com/en-us/skype-sdk/sdn/articles/installing-the-trusted-root-certificate.html

For example, you can install the certificates in the certificate keystore at the following location: *C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\Nymi\NSL*

Note: For additional information about importing certificates, see the *Nymi Enterprise Server Deployment Guide*.

Note: Ensure that you have configured the certificate settings in the nymi_agent.toml configuration file. For more information see the, *Configuration Overview* section.

Configuration Overview

Configuration settings for the `Nymi API WebSocket Interface` are contained in a toml file. If the `Nymi API WebSocket Interface` is not configured (the toml file does not exist), the `Nymi Agent` runs with the `Nymi API WebSocket Interface` disabled by default.

A sample configuration file named nymi_agent_default.toml file is included in the SDK package and is installed by default in *C:\Nymi\NymiAgent*. In order to enable the WebSocket simply make a copy of the file and name it nymi_agent.toml.

The `Nymi API WebSocket Interface` and `Nymi Agent` need to be configured:

- Each component needs a distinct TCP port (do not use 9120/tcp)
- Ensure that both components have connectivity to NES
- Ensure that there is no Network Address Translation (NAT) between the `Nymi API WebSocket Interface`, the `Nymi Agent` and the client machines.
- The components can co-locate with the NES (again ensure that distinct TCP ports are being used)

General Application Settings

These settings are application-wide and enable you to define the logging level.

- error: log only errors
- warn: log both errors and warnings
- info: log errors, warnings, and activity
- debug: log everything including debugging information

The following is the default setting: *log_level = "warn"*

Enterprise Server Settings

The *NES* section defines settings that affect the embedded Nymi-enabled Application (NEA), which is the basis `Nymi API WebSocket Interface`.

Table 2: Enterprise Service

Description	Setting
Sets the NEA name for the embedded NEA application.	`nea_name = "NymiAgent"`
The host URL for the NES server. Include only the protocol and hostname portion of the URI. For example, replace https://server.name.local with your nes_url.	`nes_url = "https://server.name.local.com"` For example, https://server.name.local.com

Description	Setting
The directory_service_id setting is called Application name in NES System Diagnostics page. To view the setting, on the NES Administrator Console page, click the **`About`** tab, then click the **`View Full System Diagnostics`** button. The setting is under the **`Nes Application Detail`** section.	`directory_service_id = "NES"`
The TLS client root CA certificate bundle to use when communicating with NES. If it is not specified a built-in bundle containing well known root CAs is used. Specify the path to a CA certificate or bundle to customize the trusted CA bundle using the PEM format.	`cacertfile = "cacertfile.pem"`

Note: * The Nymi-enabled Application expects an Nymi Bluetooth Endpoint with an endpoint name that is the based on the network interface address used to communicate with the agent. If the Nymi-enabled Application and Nymi Bluetooth Endpoint connect to the Nymi Agent on different network interfaces, the Nymi-enabled Application will not see the endpoint and will report it as missing with a status code of 5100. Nymi recommends that additional interfaces be disabled or to set a static well-known endpoint ID in the nbe.toml of the endpoint terminal and supply the endpoint ID to the Nymi-enabled Application to manually subscribe after connecting to the Nymi Agent.

Nymi API Websocket Interface Server Application Settings

The following table provides settings used to set the `Nymi API WebSocket Interface` application.

Table 3: Server Application Settings

Description	Setting
Specify the protocol supported by the `Nymi API WebSocket Interface` server. Use the default protocol `ws` to support a plain WebSocket using `ws://...` URL scheme. Use `wss` to support a secure WebSocket using TLS and the `wss://...` URL scheme. Set the protocol to `wss` in production environments.	`protocol = "wss"`
The server port to listen for `Nymi API WebSocket Interface` client WebSocket connections on. The default depends on the `protocol` settings. For the `ws` protocol the default port is 80. For the `wss` protocol the default port is 443. You can set an alternate port using this setting.	`port = 443 port = 80`
The public hostname of the server and also the subject of the server certificate.	`host = "agent.mycorp.com"`

Description	Setting
The maximum number of client connections to support. If set to zero (0), no limit is enforced.	`max_connections = 500`
The path to the CA root certificate chain in PEM format.	`cacertfile = "/path/to/certfile.pem"`
The path to the server certificate in PEM format.	`certfile = "/path/to/certfile.pem"`
The path to the server certificate private key in PEM format.	`keyfile = "/path/to/keyfile.pem"`

Note: The `Nymi Agent` must be able to receive incoming WebSocket connections on TCP port 9120 (used for communication with NBE) and on the TCP port configured for `Nymi API WebSocket Interface` connections (default 80 when using the ws protocol, and default 443 when using the wss protocol). Please ensure that these ports are open in the firewall on the server running the `Nymi Agent`.

Component Configuration Overview

The `Nymi API WebSocket Interface` and `Nymi Agent` need to be configured:

- Each component needs a distinct TCP port
- Determine how to configure transport layer security: on the server or by offloading
- Ensure that both components have connectivity to NES
- Ensure that there is no Network Address Translation (NAT) between the `Nymi API WebSocket Interface`, the `Nymi Agent` and the client machines.
- Each component can co-locate with the NES (ensure that distinct TCP ports are being used)

Nymi Component Configuration

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

The `Nymi API WebSocket Interface` is installed on each RDP client. The `Nymi API WebSocket Interface` 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.exe* file is installed.
2. Open the *C:\\Nymi\Bluetooth_Endpoint\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.

Nymi API WebSocket Interface Request, Operations and Response

The `Nymi API WebSocket Interface` contains request operations.

Requests

Request messages are received by Nymi API in JSON format as a null-terminated string argument to request() and returns a pointer to a JSON format response message as a null-terminated string from update().

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

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

Operations

Operations enable developers to interact with the NEE solution. Nymi-enabled Applications perform operations by sending request messages and receive the results of those operations as response messages. The request message payload must contain all request parameters for each operation. The response message payload contains all response parameters.

Response

Response messages are sent in response to a request message. They include the same operation and exchange field values as the original request message. Response data is in a JSON object named payload for a successful operation.

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

A response message appears in the following format:

```
{
"operation":"operation_value",
"exchange":"exchange_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. The status field is an integer value indicating the status of the operation.
- *error* always appears in the response and the value depends on the reason for the response. The error field is an empty object for successful operations and contains a descriptive error information if the operation is unsuccessful
  - 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.

Exchange Message

Nymi API 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_message* to match the requests that are sent from various threads to the responses that are received on the *update* thread.

subscribe operation

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

By default, each Nymi-enabled Application client connecting to the Nymi Agent is matched to its local endpoint based on the endpoint IP address and the connecting web application's remote location.

Note: New connections are auto-subscribed to an endpoint matching the client's remote address.

The Nymi Agent can run on the same or on a different computer as the `Nymi Bluetooth Endpoint` (NBE). When running the NBE on a thin client, such as Citrix, the NBE needs to connect to

a `Nymi Agent` running on another host. In order to accommodate this configuration, the endpoint IP of the NBE needs to be identified and communicated to the `Nymi Agent`.

`subscribe_endpoint` request operations appear in the following format:

```
{
"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. See the *Configuration Overview* for configuration information.

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

A Nymi-enabled Application 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 notification. If the requested NBE 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 Nymi 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 NBE. If the NBE becomes ready at a later time (for example, that workstation is powered on), the NEA will receive a ble_ready message at that time.

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 a presence update request is sent, the system will replay the last presence update received. When a presence state changes you will receive automatic notifications. For information about these notifications, see *presence update 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
}
}
```

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.
	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 update notification* for information about returned parameters.

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 (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 5: 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 .

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 optional if not provided it uses what is configured for the Nymi Agent. See the *Configuration Overview* .

- *query* field is a JSON object that defines the query values. Acceptable values include *NfcUID*, *Domain* and *Username*, and *NymiBandID*.
- *lookup_keys* field 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.

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.

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

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

where:

- *nes_url* field is optional if not provided it uses what is configured for the Nymi Agent. See the *Configuration Overview* .
- *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 "
  }
}
```

assert_identity Response

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 .

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: {}
}
```

Nymi API WebSocket Interface Notifications

The `Nymi API WebSocket Interface` contains response messages and responses.

Response Notifications

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.

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.

intent notification

The intent notification is used to signal an intention to take an action. For example, an intent to provide an *assert_identity* response is generated when a user taps their authorized Nymi Band against an NFC reader.

If you tap with an unauthorized Nymi Band or the device is not recognized, a non-zero state code is returned and no device ID is provided.

Intent notifications appear in the following format:

```
{
"operation": "intent",
"exchange": null,
"payload": {
   "device": "device ID",
   "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	meaning
nfc	Nymi Band is in an authenticated state and tapped an 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.

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*.

presence update notification

When Nymi API detects a change in Nymi Band presence, Nymi API generates a presence notification.

The *update* calls that you perform after you connect to the WebSocket 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",
"state":state
},
"error":{}
}
```

where:

- *proximity_value* is 0 when the Nymi Band is present and 4 when the Nymi Band is absent.
- *state_value* is one of the values in the following table.

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 6: State values for presence notifications

State Value	Definition
Absent	Nymi Band is not reachable, and the `Nymi Agent` cannot communicate with it. There may be a delay in the Nymi Agent hearing from the Nymi Band for periods of time.
Unauthenticated	Nymi Band is enrolled and worn on the user's wrist but not authenticated.
Weak	Nymi Band is in an authenticated state.

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 Bluetooth Adapter for at least 30 seconds.

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.

Status Code

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

Specific Status Codes

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

Table 7: Status codes

Status code	Error description
0	Operation completed successfully.
1000	Request made with invalid JSON.
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	Nymi Endpoint Errors. The 5000 series status codes indicate a problem communicating with, or with the state of the Nymi Bluetooth Endpoint. These errors are temporary. After communication is reestablished, and the status normalized, subscribed Nymi-enabled Application receive a ble_ready response message event.
5010	The Bluegiga BLED112 Adapter 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.
8001	NEA data is corrupt or not accessible.
8002	Missing organization name in the L1 certificate.
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

Nymi API 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 8: Log file locations

Component	Log location	Files
Nymi API	By default, the current working directory.	*nymi_api.log*
`Nymi Agent`	*C:\Nymi\NymiAgent*	*nymi_agent.log*
`Nymi Bluetooth Endpoint`	*`NBE_INSTALL_DIR`\logs* where NBE_INSTALL_DIR is the path of the Nymi Bluetooth Endpoint.	*nymi_bluetooth_endpoint.log*

Enable debug mode

When testing NAPI and builds, set the *`NYMI_DEBUG`* environment variable to any value to enable debug logging, and the restart the `Nymi Agent` and `Nymi Bluetooth Endpoint` (NBE) services.

Nymi Inc. (Nymi) believes the information in this document is accurate as of its publication date. The information is subject to change without notice.

The information in this document is provided as-is and Nymi makes no representations or warranties of any kind. This document does not provide you with any legal rights to any intellectual property in any Nymi product. You may copy and use this document for your referential purposes.

This software or hardware is developed for general use in a variety of industries and Nymi assumes no liability as a result of their use or application.Nymi, Nymi Band, and other trademarks are the property of Nymi Inc. Other trademarks may be the property of their respective owners.

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