SDK Developer Guide—WebAPI(Windows)

Nymi Connected Worker Platform 1.19.0
v1.0
2024-11-08

Contents

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 *Connected Worker Platform 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 understand and develop Nymi-enabled Applications (NEA) on Windows by utilizing the functionality of the Nymi SDK, over a WebSocket connection that is managed by a web-based or other application.

Audience

This guide provides information to Software Application Developers.

Revision history

The following table outlines the revision history for this document.

Table 1: Revision history

Version	Date	Revision history
1.0	November 08, 2024	First release of this document for the CWP 1.19.0 release.

Related documentation

- **Nymi Connected Worker Platform—Overview Guide**

  This document provides overview information about the Connected Worker Platform (CWP) solution, such as component overview, deployment options, and supporting documentation information.
- **Nymi Connected Worker Platform—Deployment Guide**

  This document provides the steps that are required to deploy the Connected Worker Platform solution.

  Separate guides are provided for authentication on iOS and Windows device.
- **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.

- **Nymi SDK Developer Guide—NymiAPI(Windows)**

  This document provides information about how to develop Nymi-enabled Applications by using the Nymi API(NAPI).
- **Connected Worker Platform with Evidian Installation and Configuration Guide**

  The Nymi Connected Worker Platform with Evidian Guides provides information about installing the Evidian components and configuration options based on your deployment. Separate guides are provided for Wearable, RFID-only, and mixed Wearable and RFID-only deployments.
- **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.
- **Nymi Connected Worker Platform with Evidian Troubleshooting Guide**

  This document provides overview information about how to troubleshoot issues that you might experience when using the Nymi solution with Evidian.
- **Nymi Connected Worker Platform—FIDO2 Deployment Guide**

  The Nymi Connected Worker Platform—FIDO2 Deployment Guide provides information about how to configure Connected Worker Platform and FIDO2 components to allow authenticated users to use the Nymi Band to perform authentication operations.
- **Connected Worker Platform with POMSnet Installation and Configuration Guide**

  The Nymi Connected Worker Platform—POMSnet Installation and Configuration Guides provides information about how to configure the Connected Worker Platform and POMSnet components to allow authenticated users to use the Nymi Band to perform authentication operations in POMSnet.
- **Nymi Band Regulatory Guide**

  This guide provides regulatory information for the Generation 3 (GEN3) Nymi Band.
- **Third-party Licenses**

  The Nymi Connected Worker Platform—Third Party Licenses Document contains information about open source applications that are used in Nymi product offerings.

How to get product help

If the Nymi software or hardware does not function as described in this document, 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 SDK Overview

The Nymi SDK provides Developers with libraries, APIs, sample code and documentation to build a Nymi-enabled Application (NEA).

Note: In this guide, the use of the term *your application*, *your web-based application*, and *your native application* refers to an NEA .

Nymi SDK delivers the Nymi API(NAPI) through a Windows Dynamically Linkable Library(DLL) named *nymi_api.dll* that developers include in a Windows application that supports a locally linked library.

Nymi WebAPI Overview

Nymi WebAPI is an RFC-6455 compliant WebSocket. NEAs, such as web-based applications use a standard WebSocket client to access Nymi WebAPI.

The Nymi WebAPI:

- Allows developers to utilize the WebSocket functionality of the Nymi SDK in a web-based or native application. The Nymi WebAPI architecture is part of the Nymi SDK.
- Provides bi-directional communication using requests/responses over a persistent connection. All messages sent and received are encoded in JSON format.
- Supports the Microsoft Windows and Apple iOS platforms only
- 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.
- Communicates with Nymi Bands over a WebSocket client and supports Nymi Band taps on a supported NFC reader or the Nymi-supplied the Bluetooth adapter. .

To enable NFC support, on the user terminal you must:

- Connect the NFC reader
- Install a compatible version of the Nymi Bluetooth Endpoint

To secure communication between Nymi Agent and WebAPI client applications, Nymi highly recommends that you enable TLS for the WebAPI interface.

When a user performs a Nymi Band tap to complete an authentication or e-signature in WebAPI application, the Nymi Bluetooth Endpoint sends an intent event that represents the tap to the application through the interface of the Nymi Agent.

Configuration parameters are set in a TOML file, as described later in this document.

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 NEA and expects to receive a pong message response. The keepalive message indicates that the connection is still responsive.

Nymi-supported web browsers send a pong message in response to the ping control frame message. The pong control frame message ensures that the session is connected to the Nymi Bluetooth Endpoint. NEA supported web browsers do not require any additional configuration to support this functionality.

If you are using a native WebSocket client, additional implementation may be required.

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

SDK Package

The SDK package contains the following folders:

- *..\nymi-sdk\windows\i686*—Contains the NAPI dll file for i686 user terminals.
- *..\nymi-sdk\windows\sampleApps*—Contains sample Nymi-enabled Applications(NEAs).
- *..\nymi-sdk\windows\x86_64*—Contains the NAPI dll file for i686 user terminals.
- *..\nymi-sdk\windows\setup\BleDriver_x64.msi*—64-bit Bluegiga driver installation file.
- *..\nymi-sdk\windows\setup\BleDriver_x86.msi* —32-bit Bluegiga driver installation file.
- *..\nymi-sdk\windows\setup\NymiRuntime-5.9.1.8.msi*—Nymi Runtime MSI installation file.
- *..\nymi-sdk\windows\setup\Nymi Runtime installer.`version`.exe* —Nymi Runtime installation file.

Sample Application

The Nymi SDK package includes a sample application that demonstrates some of the key functionality of the Nymi solution.

The sample application is a simple Javascript application that demonstrates all the basic functions that are supported by the API and allows a user to see both JSON request and response examples to help understand how the API works.

Sample Application for Nymi WebAPI

The sample application for Nymi WebAPI is located in the *..\nymi-sdk\windows\javascript\webapiSample* folder. The application prompts you for the configuration parameters that are unique to your environment.

Creating Applications with Nymi WebAPI

Customer and partner developers can use the Nymi WebAPI to develop Nymi-enabled Application (NEAs) in programming languages, such as Java or C#. The API is based on JSON messages that are exchanged with the server over a websocket connection. This chapter provides information about the supported operations.

To deploy your application, you must install the Nymi Runtime on each terminal where your application runs. The Nymi Runtime includes the following components: Nymi Bluetooth Endpoint, and Nymi Agent.

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

The Nymi Band provides authentication information about a user to applications. An application can use this information on a point-in-time basis (for simple authentication) or continuously (for both authentication and de-authentication).

Types of Nymi Band Taps

To perform an authentication task, a Nymi Band user taps their authenticated Nymi Band on either an NFC reader(NFC tap) or the Bluetooth adapter (BLE tap) that is connected to a user terminal.

The Nymi SDK allows your application to authenticate a user. A user provides their authentication intention (intent) when they perform a Nymi Band tap.

As a developer, you must decide how your application handles a Nymi Band tap. Nymi SDK provides you with three design options.

Table 2: Design Options for Nymi Band Taps

Option	Description	Benefits and Nymi Recommended Use Case
Tap and Authenticate	When a user performs an NFC tap or a BLE tap, the Nymi SDK initiates the authentication of the Nymi Band by using a cryptographic protocol over a Bluetooth connection.	Tap and Authenticate offers the best security, with a slight increase in response time. Nymi recommends that you use this design when Nymi Band users access your application from a Windows user terminal.

Option	Description	Benefits and Nymi Recommended Use Case
Authenticated Tap	When a user performs a BLE tap, NES authenticates the Nymi Band by verifying cryptographic information that the Nymi Band transmits through the BLE tap. NES does not need to establish a Bluetooth connection to perform the cryptographic operation with the Nymi Band.	Authenticated Tap offers very good security and offers a very fast response time. Nymi recommends this option when the user terminal establishes Bluetooth connections slowly, for example, when the Nymi Band user taps on the Bluetooth reader of an iOS device.
Tap and Lookup	When a user performs a tap, the NEA performs a *lookup* operation to identity the Nymi Band user. The Nymi Band and the Nymi SDK do not exchange cryptographic information, however; the Nymi Band still needs to have authenticated the user by using their fingerprint or corporate credentials authentication.	This design is a legacy option. Nymi recommends that you update your application to use either Tap and Authenticate or Authenticated Tap.

Tap and Authenticate Workflow

When a Nymi Band user taps and the Nymi-enabled Application(NEA) handles Nymi Band taps with an Tap and Authenticate design, a series of events occur before the completion of the authentication task.

The following figure provides an example of the Tap and Authenticate worflow.

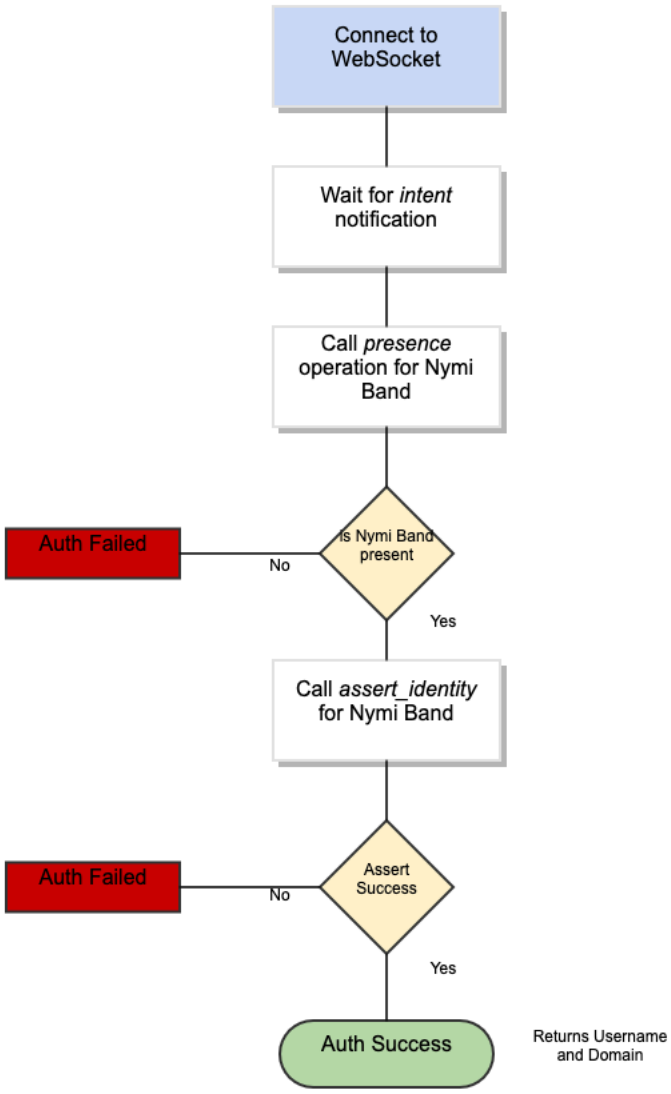

Figure 1: Tap and Authenticate Workflow

The first step is to wait for an intent notification. The intent operation tells the application that a user has placed their Nymi Band on an NFC reader that is connected to the workstation. The intent operation returns a device ID, which is the standard identifier of a Nymi Band in the CWP solution.

After the intent notification returns a device ID, the application ensures that the device is present. This action is performed in one of the following ways:

- Passively as NAPI continuously sends notifications about present Nymi Bands.
- Actively by requesting a presence operation with the desired device ID, and then waiting for a response.

For passive notifications, since NAPI sends notifications for the full list of Nymi Band present at start-up, an application can track all present bands and then check its list of current Nymi Bands. After presence is established, the application can request an assert_identity operation for the Nymi Band. The assert_identity operation uses a bi-directional challenge-response to establish a secure channel between the Nymi Agent and the requested Nymi Band. When the action results in the establishment of the secure channel, the assert_identity verifies the authentication state of the Nymi Band. When the assert_identity operation completes successfully the operation passes the username and domain of the associated user back to the application, and the application can continue with an absolute assurance that the Nymi Band is present and authenticated to the correct user.

Note: The Nymi Band exchanges data over Bluetooth Low Energy(BLE) and the exchange consists of several cryptographic operations. As a result, the assert_identity operation can take up to two seconds to complete.

Continuous monitoring of the WebSocket to watch for presence notifications indicates to an application when a user has authenticated, de-authenticated (by removing their Nymi Band), or when the user leaves a physical area. The presence notifications always returns one of the following statuses for a single Nymi Band.

- Weak—The Nymi Band is present. A strong presence is represented by the successful return of an assert_identity operation.
- Absent—The Nymi Band is not present.
- Unauthenticated—The Nymi Band is not authenticated.

Note: Ensure that the loss of presence triggers your application to log out, lock, or remove user access to functionality.

Authenticated Tap Workflow

When a Nymi Band user taps and the Nymi-enabled Application(NEA) handles Nymi Band taps with an Authenticated Tap workflow, a series of events occur that result in the NEA (web application) receiving a notification.

The notification indicates to the web application that:

- A user wearing an authenticated Nymi Band wants to perform an authentication task.
- NES has authenticated the Nymi Band over Bluetooth.

The following figure summarizes the workflow that the solution follows for an Authenticated Tap.

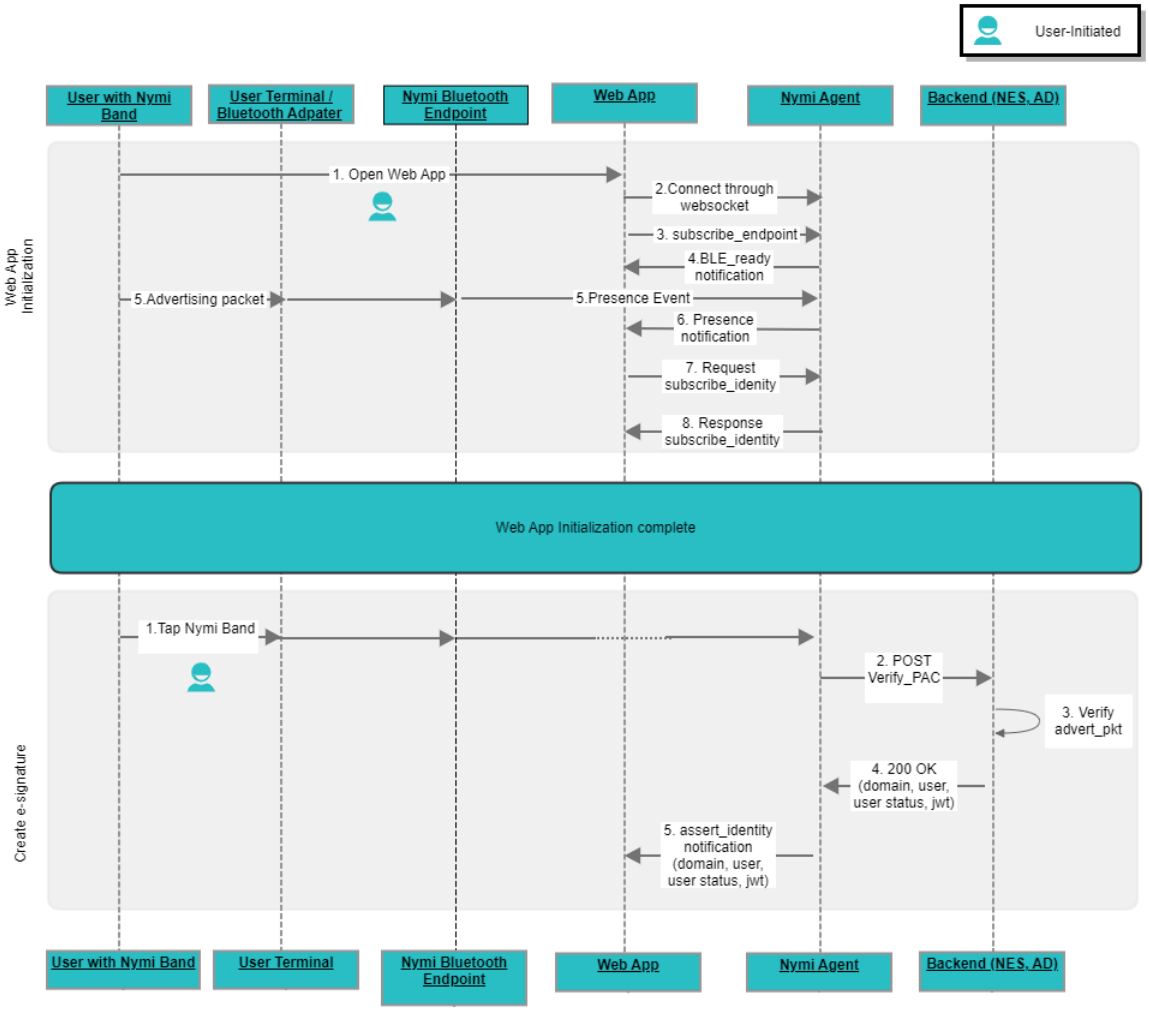

Figure 2: Workflow for Authenticated Taps

The workflow includes two distinct phases. Each phase includes user-initiated and application-initiated actions.

Phase 1—Initialize NEA

This phase occurs each time a user connects to the web application and results in the establishment of connectivity between the web application and the Nymi components.

1. User opens the NEA on their iOS device.
2. The NEA establishes a WebSocket connection to the Nymi Agent.
3. The NEA sends a *subscribe_endpoint* request to the Nymi Agent. The *subscribe_endpoint* request identifies the Nymi Bluetooth Endpoint that the web application communicates with.
4. The Nymi Agent sends a *ble_ready* notification to the web application.

5. The Nymi Bluetooth Endpoint receives advertisements from authenticated Nymi Bands that are near the Bluetooth adapter. Nymi Bluetooth Endpoint generates presence events for each Nymi Band, and sends them to Nymi Agent.
6. The Nymi Agent starts to send a *presence* notifications for each authenticated Nymi Band that is near the user terminal to the web application.
7. Web application sends a *subscribe_identity* request to the Nymi Agent.
8. Nymi Agent responds to the request. Nymi Agent returns a success response.

Phase 2—Create E-signature

This phase occurs each time a user performs an e-signature with the Nymi Band and results in the completion of an e-signature with the tap of a Nymi Band.

1. From a window within the web application the user performs an action that requires an e-signature, and then the user taps the Nymi Band on the Bluetooth adapter. Nymi Bluetooth Endpoint detects the tap and notifies to the Nymi Agent.
2. Nymi Agent request that NES verify the advertising packet of the Nymi Band.
3. NES verifies the packet and contacts Active Directory to confirm the user credentials.
4. NES returns the response to the Nymi Agent.
5. Nymi Agent sends an *assert_identity* notification to the web application. The web application reviews the notification and based on the information, completes the e-signature or does not complete the e-signature.

Tap and Lookup Workflow

When a Nymi Band user taps and the Nymi-enabled Application(NEA) handles Nymi Band taps with an Tap and Lookup design, a series of events occur before the completion of the authentication task.

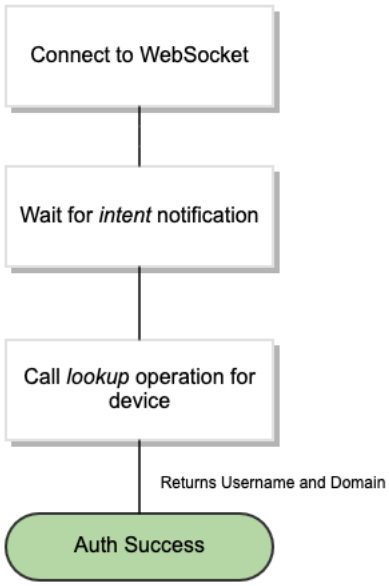

Figure 3: Tap and Lookup Workflow

The first step is to wait for an intent notification. The intent operation tells the application that a user has placed their Nymi Band on an NFC reader that is connected to the workstation. The intent operation returns a device ID, which is the standard identifier of a Nymi Band in the CWP solution.

After the intent notification, the application requests a lookup operation, which returns the username and domain of the user that is associated with the Nymi Band, then the authentication completes.

Operations and Notifications for Web App Initialization

This section summarizes the operations and notifications that initialize the Web App and allow your application to handle Nymi Band taps.

Subscribe_endpoint Operation

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

subscribe_endpoint request operations appear in the following format:

```
{
```

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

where:

- *operation* is *subscribe_endpoint*.
- *exchange* is any value and is used to match the response to the request.
- *endpoint_id* is based on the endpoint IP address. Required when the configuration uses a centralized Nymi Agent.

The *subscribe_endpoint* operation returns a status code only, no errors are returned.

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

You can only subscribe your application to one endpoint at any given time. When you request the *subscribe_endpoint* operation, the NEA is automatically unsubscribed from the previously subscribed endpoint. Any Nymi Bands that were present on the previously subscribed endpoint become absent, and your application receives 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, your application receives a *ble_ready* notification, followed by presence update notifications for any present bands on that endpoint. Otherwise, your application receives an error message. See *Bluetooth Notifications* for more information about possible error messages.

Note: The NEA remains 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, when a user turns on the user terminal), then your application receives a ble_ready*endpoint_id* message at that time.

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*.

Subscribe_identity Operation

The *subscribe_identity* operation enables your application to process Bluetooth Tap notifications through the VerifyPAC API on NES to confirm the identity of the Nymi Band user.

subscribe_identity request operations appear in the following format:

```
{
  "operation": "subscribe_identity",
  "exchange":"exchange_value",
  "payload": {

    "assertion": "jwt|none",
     "nonce": "nonce_value"
  }
}
```

where:

- *operation* is *subscribe_identity*.
- *exchange* is any value and is used to match the response to the request.
- payload can contain two optional parameters:
  - *assertion*. Acceptable values are:
    - jwt—The operation requires an NES-issued signed assertion in the form of a JSON Web Token.

  - none—The operation does not require an NES-issued signed assertion.

  When you do not specify the *assertion* parameter, the default value is none.
- *nonce* is any value that the *subscribe_identity* operation includes in the signed assertion, to prevent replay attacks. Required only when you configure the *assertion* parameter with jwt.

The *subscribe_identity* operation returns response.

The following table summarizes the status codes can appear:

Table 3: Subscribe_identity status codes

Status Code	Description
0	Connection was successfully subscribed in the WebAPI to the *subscribe_identity* workflow.
1000	*subscribe_identity* contains an invalid JSON string, or the payload contains an unacceptable assertion value.

The response appears in the following format:

```
{
  “operation”: “subscribe_identity”,
  "exchange":"exchange_value",
  “payload”: {}
  “status”: 0|1000,
  “error”: {}
}
```

When the verification succeeds, *subscribe_identity* enables the *assert_identity* notification, as described in the following section.

Intent Notification

When a user taps their authenticated Nymi Band on or near a bluetooth radio antenna or NFC reader, the action represents an intent to perform an authentication task, such as an e-signature.

Status Codes

A 2201 status code is reported when the NFC reader is unsuccessful at mapping the Nfc UID 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 contain a NymiBandID in the payload.

Presence Notifications

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

After your application establishes the websocket session, the system sends an updated notification each time any of presence parameters change.

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 same format as the presence operation.

Presence Operation (Optional)

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

When a *presence* 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 notifications.*

Presence is relative to an endpoint (the response indicates if the Nymi Band is in range of your application. 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": "NymiBandID",
     "proximity" : "proximity value",
     "service_request_state" : "service request state",
     "state" : "state"
   },
}
```

where:

- *NymiBandID*: Is the Nymi Band MAC address.
- *proximity_value*: Is 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.
- *state*: Is determined by the state of the Nymi Band; weak, absent, or unauthenticated. The following table describes the state values in more detail:

Table 4: State values for presence

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.

- *service request state*: Is 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 not zero, the Nymi Band has service level messages. If the value is '0', there are no messages

Operations for Tap and Authenticate

When a user performs an NFC tap, Nymi WebAPI provides your application with an *intent* notification and your application uses the *assert_identity* operation to verify that the Nymi Band user that taps on an NFC reader .

Assert_identity Operation

The *assert_identity* operation provides your application 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* operation 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, the operation uses the nes_url value that you specified in the Nymi Agent toml file.
- *NymiBandID* is the Nymi Band (or device) ID value that is returned in the *lookup* result.

The following code block provides an example of a JSON object that instructs Nymi WebAPI 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 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.
Active|Locked	User account is locked. This status can appear with Password Expired.

User Status	Definition
Active|PasswordExpired	User account has an expired password. This status can appear with Locked.

The last three properties can be combined into a comma 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",
     "jwt": "json web token"
  }
  "status": "0",
  "error: {}
}
```

Operations and Notifications for Authenticated Tap

The notification for an Authenticated Tap differs depending on the type of application.

For Windows and web-based iOS applications, your application uses the *subscribe_identity* operation during the initialization phase, which provides the *assert_identity* notification to verify the Nymi Band user that taps on a Bluetooth Adapter.

For native iOS applications, the Nymi Application modifies the return URL to include the JWT or error codes. Your application must decode and verify the JWT.

Assert_identity Notifications

When the user taps their Nymi Band on the Bluetooth adapter and VerifyPAC successfully verifies the user, WebAPI sends an *assert_identity* notification to the subscribed client connection with a status of 0.

You application retrieves a notification in the following format.

```
{
  "operation": "assert_identity",
```

```
  "exchange": "exchange",
  "status": 0,
  "payload": {
     "User": "username",
     "Domain": "domain_name",
     "UserStatus": "user_status"
  },
  "error ": {}
}
```

where:

- *operation* is *assert_identity*.
- *exchange* is any value and is used to match the response to the request.
- payload displays the username and domain for the Nymi Band user, and optionally the status of the user in Active Directory (AD) and the NES-issued JWT token.

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.
Active | Locked	User account is locked. This status can appear with Active and Password Expired.
Active | PasswordExpired	User account has an expired password. This status can appear with Active and Locked.

By default, NES is not configured to perform 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.

If the VerifyPAC fails, 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"
   }
 }
```

The following table summarizes the error code and error description that VerifyPAC might return to the NEA.

Table 6: VerifyPAC Errors

Error Code	Error Description
7001	*Authentication Error. Log in to the Nymi Band Application to update Nymi Band settings.*. This error appears when battery level became very low before the user charged the Nymi Band, and the real-time clock cannot keep time. To resolve this issue, the Nymi Band user must log into the Nymi Band Application while they wear their authenticated Nymi Band to reset the real time clock.
7002	*Authentication Error. Cannot find the Nymi Band in the Nymi Enterprise Server. Contact your administrator.* This error appears when the user enrolled their Nymi Band to a different NES.
7003	*Authentication Error. Key cannot be found on the Nymi Enterprise Server. Contact your administrator.* This error appears when the advertising key does not exist for the Nymi Band.
7004	*Authentication Error. Please retry.*. This error appears when the VerifyPAC operation cannot verify the authenticity of the presence authentication code (PAC). Provide the user with an message similar to the following: *Authentication Error. Retry.*
7005	*Communication error. Contact your administrator.* This error appears when there is an issue with the VerifyPAC request payload.

Note: The subscription exists until the WebSocket connection/session ends for the client.

Operations and Notifications for Tap and Lookup

When a user performs an NFC tap, Nymi WebAPI provides your application with an `intent` notification, and then your application uses the `assert_identity` and `lookup` operations verify the Nymi Band user that taps on an NFC reader .

Lookup Operation

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

- Device ID (MAC address) of the Nymi Band.

  Note: An intent notification includes the device ID or you can retrieve the device ID of a Nymi Band from NES by using the lookup operation.
- 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 7: AD user statuses

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

By default, NES does not perform 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* the NES URL.
- *query* field is a JSON object that defines the values that are passed during the request to retrieve the response. Acceptable values include *NfcUID*, *Domain* and *Username*, and *NymiBandID*.

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

Example 1

The following code block provides an example of a JSON object that instructs Nymi WebAPI 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"]
   }
}
```

Result 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 Nymi WebAPI 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"]
```

```
        }
    }
```

Result 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: {}
}
```

Troubleshooting

Nymi API writes information to log files that allow you to monitor and troubleshoot your application.

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	*C:\Nymi\Bluetooth_Endpoint \logs*	*nymi_bluetooth_endpoint.log*

Enable debug mode

When testing Nymi WebAPI 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 services.

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