Installation Guide

Advanced Authentication Device Service

Version 5.6

Legal Notices

Contents

About NetIQ Corporation

We are a global, enterprise software company, with a focus on the three persistent challenges in your environment: Change, complexity and risk—and how we can help you control them.

Our Viewpoint

Adapting to change and managing complexity and risk are nothing new

In fact, of all the challenges you face, these are perhaps the most prominent variables that deny you the control you need to securely measure, monitor, and manage your physical, virtual, and cloud computing environments.

Enabling critical business services, better and faster

We believe that providing as much control as possible to IT organizations is the only way to enable timelier and cost effective delivery of services. Persistent pressures like change and complexity will only continue to increase as organizations continue to change and the technologies needed to manage them become inherently more complex.

Our Philosophy

Selling intelligent solutions, not just software

In order to provide reliable control, we first make sure we understand the real-world scenarios in which IT organizations like yours operate—day in and day out. That's the only way we can develop practical, intelligent IT solutions that successfully yield proven, measurable results. And that's so much more rewarding than simply selling software.

Driving your success is our passion

We place your success at the heart of how we do business. From product inception to deployment, we understand that you need IT solutions that work well and integrate seamlessly with your existing investments; you need ongoing support and training post-deployment; and you need someone that is truly easy to work with—for a change. Ultimately, when you succeed, we all succeed.

Our Solutions

- Identity & Access Governance
- Access Management
- Security Management
- Systems & Application Management
- Workload Management
- Service Management

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. The documentation for this product is available on the NetIQ Web site in HTML and PDF formats on a page that does not require you to log in. If you have suggestions for documentation improvements, click **Add Comment** at the bottom of any page in the HTML version of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

NetIQ Communities, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, NetIQ Communities helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit community.netiq.com.

About this Book

The *Advanced Authentication Device Service Guide* has been designed for all users and describes system requirements that must be fulfilled before the installation of Advanced Authentication Device Service.

Intended Audience

This book provides information for individuals responsible for understanding administration concepts and implementing a secure, distributed administration model.

About Device Service

Device Service provides you with an ability to use compliant fingerprint devices, contact and contactless cards, PKI smart cards, crypto sticks, and FIDO U2F tokens during enrollment in Advanced Authentication Self-Service Portal and for further authentication.

1 System Requirements

The following table contains information about supported platforms for Device Service:

	Microsoft Windows	Apple MacOS X	Linux
Card plugin	x	x	x
FIDO U2F plugin	x	x	x
Fingerprint plugin	x		
PKI plugin	x	x	x
Bluetooth	x	x	x

Device Service for Windows supports Card and PKI redirection to Remote Desktop and Citrix terminal sessions. You must have Device Service installed on the terminal server to perform the redirection.

Device Service also supports virtual channel and you must have the Device Service installed on the both terminal client and terminal server.

NOTE: Local Admins (Windows)/ root (Mac OS X, Linux) privileges are required for installing and removing Device Service.

Ensure that the system meets the following requirements:

- Microsoft Windows 7(x64/x86) SP1/Microsoft Windows 8.1(x64/x86)/Microsoft Windows 10(x64/x86)/ Microsoft Windows Server 2008 R2/ Microsoft Windows Server 2012 R2 is installed.
- Apple MacOS X 10.11 (El Capitan).
- The following versions of Linux are installed:
    - Debian 8
    - Ubuntu 15
    - Fedora 24
    - openSUSE 42
    - SUSE Linux Enterprise Desktop 12 SP1
    - SUSE Linux Enterprise Server 12 SP1
    - Red Hat Enterprise Linux Client 7.2
    - Red Hat Enterprise Linux Server 7.2
    - CentOS 7
- One of the following browsers:
    - Microsoft Internet Explorer 11
    - Google Chrome 45 and newer
    - Mozilla Firefox 40 and newer
    - Apple Safari 8.0-9.0

  - Microsoft Edge

    To run Device Service on Microsoft Edge, perform the following steps:

    1. Open the command prompt with elevated privileges.
    2. Run the command `CheckNetIsolation LoopbackExempt -a -n=Microsoft.MicrosoftEdge_8wekyb3d8bbwe`

- Microsoft .NET Framework 4.0 is installed for fingerprint or set the parameter `fingerprint.isoSupported` to `false`
- Only Bluetooth is supported, BLE is not supported

NOTE: It is not recommended to use the Bluetooth functionality on VMware virtual machines, because false authentication can occur when Bluetooth device is disabled or it is out of range.

For more information about additional system requirements, see the following sections:

- Supported Card Readers and Cards
- Supported Devices for PKI
- Supported Fingerprint Readers

1.1 Supported Card Readers and Cards

Advanced Authentication stores the serial number of a card during enrollment and validates the serial number later during the user's authentication.

Advanced Authentication supports the following cards and card readers:

- **Contactless card readers**
  - ACS ACR122
  - Broadcom Corp Contactless SmartCard
  - Elatec RFID
  - HID OMNIKEY CardMan 5x25
  - HID OMNIKEY 5326
  - HID OMNIKEY 5x2x
  - LEGIC LE-762-1N (supported only for Microsoft Windows and requires installation with specific parameters and disabling of other card plug-ins), supported with only card.smarfidManualMode=true
  - LEGIC LM3000 (supported only for Microsoft Windows and requires installation with specific parameters and disabling of other card plug-ins)
  - RFIDeas pcProx series (supported only for Microsoft Windows)
  - NXP PR533
- **Non supported readers**
  - LEGIC AIR ID series
- **Contactless smart cards**
  - HID iClass series
  - HID Prox series
  - MIFARE Classic 1K/4K, Ultra Light, Ultra Light C, Plus
  - MIFARE DESFIRE 0.6, MIFARE DESFIRE EV1, MIFARE SE, DESFire

1.2 Supported Devices for PKI

Advanced Authentication supports the certificate-based PKCS#11 contact smart cards and USB tokens (crypto sticks).

Device Service supports the following devices:

- Aladdin eToken PRO 32k/72k with SafeNet Authentication Client 9
- ruToken

To use PKI, specify a PKCS#11 module for your PKI device. See PKI Settings for more information.

The following are the requirements for used certificates:

1. Certificate must contain the OCSP or CRL link to check revocation status.
2. Certificate must contain a key pair: public and private key in the x509 format. The certificates that do not comply with the requirements are ignored (hidden during enrollment).

NOTE: The cards Cosmo polIC 64K V5.2 and Cyberflex Access 64K V1 SM 2.1 support the certificate-based enrollment only (generate a key pair mode is not supported).

1.3 Supported Fingerprint Readers

Device Service supports fingerprint readers that use Windows Biometric Framework (WBF), Lumidigm readers, and Digital Persona readers.

Ensure that the system meets the following requirements for the WBF compliant readers:

- A reader must be available in Device Manager in the **Biometric devices** section.
- The **Windows Biometric Service** (in services.msc) must be set to `Automatic` and must be in a running state.
- The policies **Allow to use of biometrics, Allow users to log on using biometrics, Allow domain users to log on using biometrics** (Computer Configuration - Administrative Templates - Windows Components - Biometrics) must be enabled.

Device Service supports the following fingerprint readers:

- Lumidigm readers
- Digital Persona readers
- NEXT Biometrics NB-3010-UL
- Precise Biometrics 100 X with AuthenTec AES2501B
- Zvetco Verifi P2500 with AuthenTec AES2550
- Zvetco Verifi P5100
- Zvetco Verifi P5200 with TouchChip Fingerprint Coprocessor
- Zvetco Verifi P6000
- Synaptic FP Sensors (WBF) (VID=138A, PID=0011)
- Synaptic FP Sensors (WBF) (VID=138A, PID=0017)
- Validity Sensor (VFS495) (VID=138A, PID=003F)

- Validity Sensors (WBF) (VID=138A, PID=0050)
- SecuGen Hamster Plus (HSDU03P)

Device Service does not support the following devices:

- SecuGen Hamster IV (HFDU04)
- SecuGen Hamster (HFDU02R)
- Synaptics WBDI (Lenovo t460s laptops)
- Futronic FS80, FS88

Usage of fingerprint readers requires manual configuration. For more information, see Fingerprint Settings.

NOTE: Swipe readers may face issues with fingerprint matching because of low quality sensors.

2 Installing and Uninstalling Device Service

Before installing Device Service, ensure that you close all the web browsers. The installation procedure varies for different operating systems.

NOTE: You can find the Device Service component in the Advanced Authentication Enterprise Edition or the Remote Access Edition distributive package.

Device Service on Microsoft Windows

- Installing Device Service on Windows
- Uninstalling Device Service on Windows

Device Service on Apple Mac OS X

- Installing Device Service on Mac
- Uninstalling Device Service on Mac

Device Service on Linux

- Installing Device Service on Linux
- Upgrading Device Service on Linux
- Uninstalling Device Service on Linux

WARNING: During the upgrade of Device Service on Apple Mac OS X and Linux, the configuration file is overwritten with a default one. Ensure that you have a copy of the file and put it back to the folder after the Device Service upgrade.

2.1 Installing Device Service on Windows

1. Run `naaf-deviceservice-x86-release-<version>.msi`.

   IMPORTANT: For LEGIC readers, you need to install Device Service by running the command line:

   ```
   msiexec /i naaf-winclient-x86-release-<version>.msi TOKEN="XXX" KEY="YYY"
   ```

   `XXX` - Token value (HEX <= 12 byte)

   `YYY` - 3Des Key (HEX 16 byte)

   If you leave the `TOKEN/KEY` parameters blank or enter invalid commands, Device Service does not detect the LEGIC reader.

2. Click **Next**.
3. Read and accept the licence agreement.

4. Click **Next**.
    - To change the destination folder, click **Change** and select an applicable destination.
    - To continue, click **Next**.
5. Click **Install** and wait until the component is installed.
6. Click **Finish**.

> **NOTE:** To upgrade Device Service 5.3 and 5.4 on a Windows machine that has a McAfee virus protection software installed, ensure to disable the McAfee protection. For more information about how to disable McAfee protection for a temporary period, see link1 and link2.

2.2 Uninstalling Device Service on Windows

You can uninstall Device Service through the Setup Wizard or through Control Panel.

- Uninstalling Device Service through Setup Wizard
- Uninstalling Device Service through Control Panel

2.2.1 Uninstalling Device Service through Setup Wizard

1. Run `naaf-deviceservice-x86-release-<version>.msi`.
2. Click **Next**.
3. Select **Remove** and click **Next**.
4. Click **Remove**.

2.2.2 Uninstalling Device Service through Control Panel

To uninstall Device Service through Control Panel, select one of the following options that corresponds to your operating system:

- Microsoft Windows 7
- Microsoft Windows 8.1
- Microsoft Windows 10

Microsoft Windows 7

1. In the **Start** menu, select **Control panel** and then double-click **Programs and Features**.
2. Select **NetIQ Device Service** and click **Uninstall**.
3. Confirm the uninstallation.

Microsoft Windows 8.1

1. In the **Search** menu, select **Apps > Control Panel > Programs > Programs and Features.**
2. Select **NetIQ Device Service** and click **Uninstall**.
3. Confirm the uninstallation.

Microsoft Windows 10

1. Right-click **Start** and select **Control Panel > Programs > Programs and Features**.
2. Select **NetIQ Device Service** and click **Uninstall**.
3. Confirm the uninstallation.

2.3 Installing Device Service on Linux

IMPORTANT: To use Device Service for FIDO U2F tokens, you must allow the FIDO U2F usage on Linux. For more information, see yubico FAQ.

To install Device Service on Linux operating system, run the following commands depending on your platform.

Ubuntu, Debian (deb package)

The components libnss3-tools are for Card/PKI plug-in and `bluez` for Bluetooth plug-in.

```
sudo apt-get install libnss3-tools
sudo apt-get install bluez
sudo dpkg -i naaf-deviceservice-linux64-release-<version>.deb
```

openSUSE, SUSE

The components nss-tools are for Card/PKI plug-in and `bluez` for Bluetooth plug-in.

```
sudo zypper install mozilla-nss-tools
sudo zypper install bluez
sudo rpm -i naaf-deviceservice-linux64-release-<version>.rpm
```

Fedora or CentOS

The components nss-tools are for Card/PKI plug-in and `bluez` for Bluetooth plug-in.

```
sudo yum install nss-tools
sudo yum install bluez
sudo rpm -Uvh naaf-deviceservice-linux64-release-<version>.rpm
```

NOTE: During the installation of Device Service on RHEL operating system, there could be dependency issues related with the pcsc-lite package. Install the required package with **yum install pcsc-lite** and restart the installation of Device Service.

2.4 Upgrading Device Service on Linux

To upgrade Device Service on Linux operating system, run the following commands depending on your platform.

NOTE: Device Service has been renamed from **deviceservice** to **naaf-deviceservice** from Advanced Authentication 5.3 Hotfix 1.

Ubuntu, Debian (deb package)

To upgrade Device Service 5.3 or later, remove the old package and install a new package.

1. Remove device service package.

   ```
   sudo apt-get remove deviceservice-<version>.x86_64

   sudo apt-get install bluez

   sudo dpkg -i naaf-deviceservice-linux64-release-<version>.deb
   ```

openSUSE, Fedora (rpm package)

To upgrade Device Service 5.3 or later, remove the old package and install a new package.

openSUSE

1. Remove device service package.

   ```
   sudo rpm -e deviceservice-<version>.x86_64

   sudo zypper install bluez

   sudo rpm -i naaf-deviceservice-linux64-release-<version>.rpm
   ```

Fedora

1. Remove device service package.

   ```
   sudo rpm -e deviceservice-<version>.x86_64

   sudo yum install bluez

   sudo rpm -Uvh naaf-deviceservice-linux64-release-<version>.rpm
   ```

2.5 Uninstalling Device Service on Linux

Run the following commands depending on your platform:

Ubuntu, Debian (deb package)

```
sudo dpkg --purge naaf-deviceservice-<version>.x86_64
```

openSUSE, Fedora

```
rpm -e naaf-deviceservice-<version>.x86_64
```

2.6 Installing Device Service on Mac

1. Run `naaf-deviceservice-macos-release-<version>.pkg`.
2. Click the Apple icon in the top-left corner and select **System Preferences.**
3. Click the **Security & Privacy** icon.
4. Click **Open Anyway** on the **General** tab.
5. Click **Continue**.

6. Read and accept the licence agreement.
7. Select the disk where you want to install Device Service and click **Continue**.
8. Click **Install.**
9. Specify the root account credentials and click **Install Software.**
10. Click **Close**.

2.7 Uninstalling Device Service on Mac

Delete the folder `DeviceService in /Library/LaunchDaemons/NetIQ/` to uninstall Device Service on Mac.

3 Configuring Device Service

Device Service contains the configuration file that is located in the following folder, depending on your platform:

- **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties.`
- **Linux**: `/opt/NetIQ/Device Service/config.properties.`
- **Apple Mac OS X**: `/Library/LaunchDaemons/NetIQ/Device Service/config.properties.`

WARNING: During the upgrade of Device Service on Apple Mac OS X and Linux, the configuration file is overwritten with a default one. Ensure that you have a copy of the file and put it back to the folder after the Device Service upgrade.

NOTE: In the `host.ports` parameter, the supported ports are 8440, 8441, and 8442.

See the following settings for the Device Service configuration.

- Section 3.1, “Card Settings,” on page 19
- Section 3.2, “Fingerprint Settings,” on page 20
- Section 3.3, “PKI Settings,” on page 21
- Section 3.4, “Performing Bulk Replacement of Configuration File,” on page 23

To apply the changes, reboot the machine.

3.1 Card Settings

Advanced Authentication supports the Microsoft policy Interactive logon: Smart card removal behavior, which allows to select an action on a card event. You can configure it to perform a force log off or lock a user session when a user presents card to the reader.

To use LEGIC LM3000 or LEGIC LE-762-1N readers, you must disable the other card plug-ins to avoid conflicts. To do this, perform the following steps:

1. Open the configuration file depending on the platform:
    - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`.
    - **Linux**: LEGIC and RFIDeas readers are not supported.
    - **Apple Mac OS X:** LEGIC and RFIDeas readers are not supported.
2. Change the existing parameters based on the following scheme:
    - `card.omnikeyEnabled: false`
    - `card.rfideasEnabled: false`
    - `card.smarfidEnabled: true`
    - `card.desfireEnabled: false`

  - `card.isCardIdGenerated=true` to generate a new card identifier during enrollment. The default value is false and during each enrollment, the card identifier is not changed. The feature can be used only for LEGIC readers.
  - `card.smarfidManualMode=true` Without the `card.smarfidManualMode` in the config file or when `card.smarfidManualMode=false`, the reader's LED is blue (read mode) by default and it always starts to blink when you put a card on it.

    When `card.smarfidManualMode=true` the reader's LED is green (ready mode) by default and does not blink when you put a card on the reader. It will blink only if you are on Logon/Unlock screen and Windows Client requests to put a card. 1:N has to be disabled to disable auto-waiting for a card for Logon/Unlock screen. For more information on disabling 1;N, refer to Disabling 1:N. Also **Interactive logon: Smart card removal behavior** policy must be disabled to disable auto-waiting for a card when a user is logged in. For more information on disabling **Smart card removal behavior** policy, refer to the link. You can use the feature only for LEGIC readers.
  - `card.smarfidManualBeepEnabled=true`. You can use this option only when the manual mode is enabled (`card.smarfidManualMode=true`). When you set card.smarfidManualBeepEnabled to `true`, you can hear beeps from a supported LEGIC reader when you put a card on it. The default value of the parameter is `false` and the beeps are muted.
3. Save the changes.
4. Restart the workstation.

3.2 Fingerprint Settings

NOTE: You must install Microsoft .NET Framework 4.0 for using fingerprint.

Device Service supports the following modes for fingerprint readers:

- **fingerprint.mode: 1** to use the WBF API mode: In this mode, Advanced Authentication works with a processed fingerprint reader in Windows Biometric Framework API.
- **fingerprint.mode: 2** to use the WBF Direct mode: In this mode, Advanced Authentication works directly with a device driver. This is the default mode.

  NOTE: Some WBF compliant readers may work only in the WBF Direct mode, for example, the NEXT Biometrics readers. You can download the NEXT Biometrics driver from the link.
- **fingerprint.mode: 3** to use the Lumidigm mode. You must install the Lumidigm Drivers. You can download the drivers from the HID Global website. Some devices require that the Lumidigm Device Service is installed.
- **fingerprint.mode: 4** to use the DigitalPersona mode. You must install the DigitalPersona U.are.U RTE. You can download it from the DigitalPersona website.

To change the finger print settings, perform the following steps:

1. Open the configuration file depending on your platform:
   - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`.
   - **Linux**: Fingerprint readers are not supported.
   - **Apple Mac OS X:** Fingerprint readers are not supported.
2. Add a string that specifies a mode. For example, **fingerprint.mode: 3** to use the Lumidigm mode.

3. Add optional parameters (if required):
    - `fingerprint.captureTimeout: 15` of capture inactivity in seconds.

    NOTE: The parameters are case-sensitive.

    ---

4. Save the changes.
5. Restart your machine.

NOTE: The parameter `fingerprint.isoSupported: true` (default value is `true`) helps Device Service to extract ISO from raw image that it gets from user who scanned his fingerprint for authentication. This parameter helps to eliminate this additional step on the server and improves the authentication speed on the server.

If you set the parameter to `false`, Device Service sends raw image to Advanced Authentication server and the server will need to extract ISO to compare it with a stored authenticator. This may cause performance issues in environments where hundreds of users perform fingerprint authentication at the same time.

3.3 PKI Settings

To use PKI, you must specify a PKCS#11 module for your PKI device. To do this, perform the following steps:

1. Open a configuration file depending on your platform:
    - **Microsoft Windows**: `C:\ProgramData\NetIQ\Device Service\config.properties`.
    - **Linux**: `/opt/NetIQ/Device Service/config.properties`.
    - **Apple Mac OS X**: `/Library/LaunchDaemons/NetIQ/Device Service/config.properties`.
2. Remove the hash sign(`#`) before `vendorModule` to remove any comments from the parameter.
3. Specify a path to a PKCS#11 module.
    - **Microsoft Windows**:
        - **for eToken PRO:** `pki.vendorModule: eToken.dll`.
        - **for ruToken**: `pki.vendorModule: rtPKCS11.dll`.

    NOTE: You can specify more than one PKCS#11 library with semicolon in the format: `pki.vendorModule: eToken.dll;rtPKCS11.dll`

    If a vendor module is located out of the **system32** folder, use `\\`. The quotation marks are not needed even if there are spaces in the path. For example, `pki.vendorModule: C:\\Program Files\\ActivIdentity\\ActivClient\\acpkcs211.dll`.

    - **Linux**:
        - **for eToken PRO**: `pki.vendorModule: /usr/lib/libeTPkcs11.so`.
    - **Mac OS X**:
        - **for eToken PRO**: `pki.vendorModule: libeTPkcs11.dylib`.

    You can find a list of the known PKI modules from the link.

NOTE: If you have specified some `pki.vendorModules` separated by a semicolon, you must specify the same number of values for `pki.blockingMode`. For example, `pki.blockingMode: true;false.`

PKI plugin of the Device Service supports the automatic mode, where the known vendor modules are detected automatically. You must specify: `pki.vendorModule: auto`.

The following are the auto detectable vendor modules for different platforms.

Microsoft Windows

- `rtPKCS11.dll, the default pki.blockingMode: true`
- `eToken.dll, the default pki.blockingMode: true`
- `acpkcs211.dll, the default pki.blockingMode: false`

Linux

- `libeToken.so, the default pki.blockingMode: true`

Mac OS

- `libeToken.dylib, the default pki.blockingMode: true`

4. Specify the optional parameters (if required):
   a. **Hash method**

      pki.`hashMethod: SHA256`

      The default value is `SHA256` and you can specify this value, if a parameter is not presented. The following methods are also supported: `SHA224, SHA384, SHA512, RIPEMD160`. To set the methods, ensure that the PKCS#11 module supports the required hash method.

   b. **Padding**

      `pki.padding: PKCS#1`

      The default value is PKCS#1 and you can specify this value, if a parameter is not presented.The following options are also supported: **PSS, OAEP**.

   c. **Key size**

      `pki.modulusBits: 2048`

      The default value is 2048 bit. For example, eToken PRO 32k does not support it and you need to set 1024 to use it.

   d. **Blocking mode**

      pki.`blockingMode: true`

      The default value is `True`. OpenSC does not support the 'waiting for card' mechanism completely and it requires to change the option to `False`. Most of the vendors should work fine with the default mode.

      NOTE: If you specify both the parameters `pki.vendorModule: auto` and `pki.blockingMode`, the `pki.blockingMode` does not overwrite a blocking mode that is pre-defined for an autodetectable vendor module.

      ---

5. Save the changes.
6. Restart the workstation.

3.4 Performing Bulk Replacement of Configuration File

To customize configuration of Device Service on multiple computers in domain, perform the following instructions:

1. Create a configuration file `config.properties` with the required parameters.
2. Copy this configuration file on a network folder.
3. Open **Group Policy Management** console.
4. Right-click the domain name and select **Create GPO in this domain, and Link it here.**
5. Specify a name for the **Group Policy Object**. It is used to update the Device Service configuration file. Click **OK**.
6. Right-click the created GPO and click **Edit**.
7. Browse **Computer Configuration > Preferences > Windows Settings**.
8. Right-click **Files** and select **New > File**.
9. Change **Action** to **Replace**.
10. In **Source file(s)** specify the full path of the configuration file located on the network folder.
11. In **Destination File**, specify the path: `C:\ProgramData\NetIQ\Device Service\config.properties.`
12. Clear all the **Attributes** options.
13. Click **OK**.
14. Create a group in the domain that contains computers on which you want to replace the Device Service configuration file.
15. In the **Security Filtering** section of the **Group Policy Management** console, for the used GPO remove the **Authenticated Users**.
16. Click **Add** and select the created group.
17. Click **Delegation**.
18. Right-click the added group and select **Edit settings**, **delete**, **modify security.**
19. Run **gpupdate /force** on the computer where you will replace the configuration file or wait till the policy is automatically applied.

4 Troubleshooting

This chapter provides information about troubleshooting Device Service.

- Section 4.1, “Generic Issues,” on page 26
- Section 4.2, “Card Related Issues,” on page 27
- Section 4.3, “FIDO U2F Related Issues,” on page 27
- Section 4.4, “Fingerprint Related Issues,” on page 27
- Section 4.5, “PKI Related Issues,” on page 27
- Section 4.6, “Bluetooth Issues,” on page 28

To investigate the possible issues, you may be asked to provide the debug logs. The following information helps you to enable logging on different platforms.

Microsoft Windows

To enable debug logging for all Client components, follow the steps:

1. Run `DiagTool.exe` (the tool must have Microsoft .NET Framework 3.5 installed).
2. Click **Clear All** (if applicable) in the **Debug logs** tab.
3. Click **Enable**.
4. Restart the machine.
5. Reproduce your problem.
6. Run `DiagTool.exe`.
7. Click **Save** logs in the **Debug logs** tab.
8. Specify a file name and path. Click **Save** to save the logs.
9. Click **Disable** to disable the logging.
10. Click **Clear All**.

If you do not have the Diagnostic Tool, you can perform the steps manually:

1. Create a text file `C:\ProgramData\NetIQ\Logging\config.properties`.
2. Add a string to the file: `logEnabled=True` that ends by a line break.
3. Create a directory: `C:\ProgramData\NetIQ\Logging\Logs\`.
4. Restart the workstation.
5. Reproduce your problem.
6. Pack the logs located in `C:\ProgramData\NetIQ\Logging\Logs\` into a zip package.
7. Change `logEnabled=True` to `logEnabled=False` in `C:\ProgramData\NetIQ\Logging\config.properties`.

Apple Mac OS X

To enable logging for the component, perform the following steps:

1. Create a text file `/Library/LaunchDaemons/NetIQ/Logging/config.properties`.
2. Add a string to the file: `logEnabled=True` that ends by line break.
3. Save changes.
4. Create a `Logs` folder in `/Library/LaunchDaemons/NetIQ/Logging/`.
5. Stop the service by running the command in the terminal: `sudo launchctl unload /Library/LaunchDaemons/com.netiq.deviceservice.plist`.
6. Start the service: `sudo launchctl load /Library/LaunchDaemons/com.netiq.deviceservice.plist`

Linux

To enable logging for the component, perform the following steps:

1. Create a text file `/opt/NetIQ/Logging/config.properties`.
2. Add a string to the file: `logEnabled=True` that ends by line break.
3. Save changes.
4. Create a `Logs` folder in `/opt/NetIQ/Logging/`.
5. Stop the service by running the command in the terminal: `sudo service deviceservice stop.`
6. Start the service: `sudo service deviceservice start`.

Logs are generated in the `/opt/NetIQ/Logging/Logs` directory.

4.1 Generic Issues

These issues could happen with any service such as Bluetooth, PKI, Fido or Fingerprint.

After you install a new browser and then try to enroll or test authenticator, an error message `Service is not available` is displayed.

The root cause for this issue is, device service keeps the certificates for itself during installation. So if the browser is installed after installing the device service, the browser will not have the required certificates.

To fix the issue, open a new browser window and access one of following URLs. Depending on the method used, apply the appropriate certificate.

- https://127.0.0.1:8440/api/v1/card/getmessage?nowait
- https://127.0.0.1:8441/api/v1/fidou2f/abort
- https://127.0.0.1:8442/api/v1/fingerprint/capture
- https://127.0.0.1:8440/api/v1/pki/getmessage?nowait
- https://127.0.0.1:8440/api/v1/bluetooth/getdevices

4.2 Card Related Issues

To troubleshoot the Card related issues you can check the link: https://127.0.0.1:8440/api/v1/card/getmessage?nowait.

The response format is as follows:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following status is implemented:

- `NO_READER`: Indicates that the card service did not detect a card reader connected.
- `READER_ON`: Indicates that the card service detected a card reader connected.
- `NO_CARD`: Indicates that there is no card on the reader.
- `CARD_ON`: Indicates that a card is presented to the reader.

NOTE: Card ID can be used only with `CARD_ON` and `NO_CARD` status.

4.3 FIDO U2F Related Issues

To troubleshoot the FIDO U2F related issues, see: https://127.0.0.1:8441/api/v1/fidou2f/abort. The service should return: { "result":"ok" } when a FIDO U2F token is connected.

4.4 Fingerprint Related Issues

To troubleshoot the fingerprint related issues, see: https://127.0.0.1:8442/api/v1/fingerprint/capture. Open the URL while you are presenting your finger on the reader.

The following fields are included in the output:

- captureStatus: Can be '`Ok`', '`Timeout`', '`Error`', '`NoReader`'.
- Width, Height: Fingerprint image size (in pixels).
- Dpi: Dots per inch (used on matching side).
- BitsPerPixel: Bits per pixel (usually 8 bits).
- BytesPerLine: Bytes per one line in image (include align).
- Image: Fingerprint image encoded using base-64 in gray scale.

An example of a sample output:
{`"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height":360,"Image":"<fingerprintdata>","Width":256,"captureStatus":"Ok"`}.

4.5 PKI Related Issues

To troubleshoot the PKI related issues you can check the URL: https://127.0.0.1:8440/api/v1/pki/getmessage?nowait.

The service returns:

- `NO_READER` if no reader is connected.
- `NO_CARD` if a card is not presented.
- `CARD_ON` if a card is presented.

4.6 Bluetooth Issues

To troubleshoot the Bluetooth related issues, refer to the following URL: https://127.0.0.1:8440/api/v1/bluetooth/getdevices. It returns a list of the Bluetooth devices that have been discovered.

For more information on the Bluetooth, see “Bluetooth Plug-in” in the *Chapter 5, “Developer Information,” on page 29*.

5 Developer Information

Currently the supported opened ports are 8440, 8441, 8442 but it is better to use 8440, as other ports may be deprecated in the future releases.

5.1 Card Plug-in

To check the Card Service you may open the following URL: https://127.0.0.1:8440/api/v1/card/getmessage?nowait.

The response format:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following statuses are implemented:

- NO_READER means that the Card service didn't detect a card reader connected,
- READER_ON means that the Card service detected a card reader connected,
- NO_CARD means that there is no card on the reader,
- CARD_ON means that a card is presented to the reader.

NOTE: cardid is used only with CARD_ON and NO_CARD statuses.

Examples of commands:

- https://127.0.0.1:8440/api/v1/card/getmessage?nowait - immediately returns a current status. Possible values [NO_READER, NO_CARD, CARD_ON]
- https://127.0.0.1:8440/api/v1/card/getmessage?wait - waits for a next event (e.g. card presented or card removed)

  NOTE: When you disconnect the reader with a card on, two messages will arrive: NO_CARD, NO_READER. But the first one will be caught with getmessage?wait.When you plug in a reader with a card on, there will be the two events: READER_ON, CARD_ON. And as a result READER_ON will be returned.

- https://127.0.0.1:8440/api/v1/card/getreaderon?nowait - immediately returns READER_ON if a reader is attached and NO_READER otherwise.

- https://127.0.0.1:8440/api/v1/card/getreaderon?wait - immediately returns READER_ON if a reader is attached or waits till it's attached
- https://127.0.0.1:8440/api/v1/card/getcardon?nowait - immediately returns NO_READER if a reader isn't attached, NO_CARD if a card isn't presented or CARD_ON if a card is presented
- https://127.0.0.1:8440/api/v1/card/getcardon?wait - immediately returns NO_READER if a reader isn't attached or wait till the card will be presented on a reader.

  NOTE: It will wait the next tap of a card even if a card is already on a reader.

- https://127.0.0.1:8440/api/v1/card/getcardoff?nowait&cardid=<cardid> - immediately returns NO_READER if a reader isn't attached, NO_CARD if a card isn't presented on the reader or CARD_ON if a card is presented on the reader. Use cardid to wait when a specific card is removed.
- https://127.0.0.1:8440/api/v1/card/getcardoff?wait - returns immediately with NO_READER if a reader isn't attached. If there is no card presented on a reader, it returns NO_CARD immediately else waits till the card is removed from the reader
- https://127.0.0.1:8440/api/abort?cancel-cookie=xxx - all of the "wait" methods support cancel-cookie=xxx parameter.E.g. https://127.0.0.1:8440/api/v1/card/getmessage?wait&cancel-cookie=xxx.And by calling abort with a cancel-cookie, all waiting methods with the same specfied cookie are terminated.

5.2 FIDO U2F Plug-in

To check the FIDO U2F Service you may open the following URL: https://127.0.0.1:8441/api/v1/fidou2f/abort. The service should return: { "result":"ok" } when a FIDO U2F token is connected.

Available methods

FIDO U2F Service provides the following POST-methods:

https://127.0.0.1:8441/api/v1/fidou2f/sign - Performs the U2F Authenticate operation.

```
{
"signRequests":
[
{"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-
ODg09vpp62P6owh4qmQ",
"appId":"https://demo.yubico.com"
}
]
}
```

https://127.0.0.1:8441/api/v1/fidou2f/register - Performs the U2F Register operation.

```
{
"registerRequests":
[
{"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"appId":"https://demo.yubico.com"
}
],
"signRequests":[]
}
```

signRequest can be empty, or contain serial of for the key handle validation

```
{
"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-
ODg09vpp62P6owh4qmQ",
"appId":"https://demo.yubico.com"
}
```

In case of success both methods above returns JSON reply in the U2F specification format:

or an error:

```
{ "errorCode"=1, "errorMessage"="Error Text"}
```

where:

errorCode - error code

errorMessage - additional error text

errorCode description:

1. Device other error. If the token is missing, errorMessage contains "Please connect a U2F token."
2. Device bad request. The visited URL doesn't match the App ID or not using HTTPS
3. Configuration unsupported
4. Token is not registers - for authentication process or token already registered - for register process, to enable this check, specify "signRequests" in the body of the register request).
5. Timeout - no answer from token. (if the user didn't press a button within a given timeout)

And the following GET-methods:

https://127.0.0.1:8441/api/v1/fidou2f/abort - Aborts all pending operations

5.3 Fingerprint Plug-in

To check the WBF Capture Service you may open the following URL: https://127.0.0.1:8442/api/v1/fingerprint/capture. Present your finger on the reader while the URL is loading.

The following fields are included into the output:

- captureStatus - can be 'Ok', 'Timeout', 'Error', 'NoReader'.
- Width, Height - fingerprint image size (in pixels).
- Dpi - dots per inch (used on matching side).
- BitsPerPixel - bits per pixel (usually 8 bits).

- BytesPerLine - bytes per one line in image (include align).
- Image - fingerprint image encoded using base-64 in gray scale.

E.g.
{"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height":360,"Image":"<fingerprintdata>","Width":256, "captureStatus":"Ok"}.

5.4 PKI Plug-in

PKI plug-in supports the following options:

- `vendorModule=eTPKCS11.dll` - PKCS#11 implementation library of a needed vendor.
- `hash=SHA1 or SHA224, SHA256` (this is a default value if not presented), SHA384, SHA512, RIPEMD160.
- `padding=PKCS#1` (this is a default value if not presented) or PSS, OAEP.
- `modulusBits=2048` - key size (this is a default value if not presented). E.g. eToken PRO 32k doesn't support it and you need to set 1024 to use it.
- `blockingMode=True`. The default value is True. OpenSC supports the 'waiting for card' mechanism not completely and it requires to change the option to False. The most of vendors should work fine with the default mode.

PKI plugin uses the simulatar API for card / token detection and two new POST methods pki/enroll, pki/login:

Available methods:

Card service provides the following POST-methods

- `https://127.0.0.1:8440/api/v1/pki/getcertificates` - GET method to get all certificates from a token

```
{ "readerid"=0, "certificates" : [{
"keypairid":"9beb","certificate":"30820371308202daa00....0b90d7290a1a76b0450264dd5
36d2cb057230f8dbfa8cfda05"}] }
```

slotid - slot ID

keypairid - id of the key pair in the certificate. Save it and use later for future logon operations.

certificate - certificate value in DER format.

```
- https://127.0.0.1:8440/api/v1/pki/generatekeypair- POST method, Request Body:
{"pin":"your_pin"}
```

// Replace with your token pin or empty if there is no pin

```
{ "readerid"=your_reader_id, "keypairid":"6f4712e554544ac3",
"modulus":"a1709fb049c35fdc6695193e9dd980c713c....91daaa9d2604eeeaad73d13b1",
"exponent":"010001"}
```

keypairid - id of the key pair in the certificate. save it and use later for future logon operations.

modulus - modulus

exponent - big exponent

```
- https://127.0.0.1:8440/api/v1/pki/signchallenge - POST method, Request Body:
{"challenge":"3128", "pin":"your_pin", "keypairid":"9beb" }
```

challenge in hex-string format(even length, sice one byte is two hex symbols)

pin - pin to the token

keypairid - id of the keypair from token, you can get it from previous enroll operation

in case of success it returns signature for the given challenge in the hex format{ `"readerid"=your_reader_id, "hash":"SHA1", "padding":"PKCS#1", "signature":"58ad84f3a9b7244031aa55c0d0ad753b1a480ae709a37210d48....493130d7b11f128ea2be1fcc42d123bdb715a153974e992b16d022" }`

hash - used hash method

padding - used padding

- https://127.0.0.1:8440/api/v1/pki/verifychallenge - POST method, Request Body {"challenge":"3128", "pin":"your_pin", "keypairid":"9beb", "signature":"58ad84f3a9b72....bdb715a153974e992b16d022" }

in case of an error two methods above returns an error:

{ `"errorCode"="ERROR_ID"`}

Possible values of ERROR_ID:

`PLUGIN_NOT_INITTED` - not initted library, etc. dll was not provided

`METHOD_NOT_FOUND` - method not found

`NO_CARD` - no token or no card are presented. Use wait methods to get an event.

`JSON_PARSE_FAILED` - bad request body

`WRONG_PIN`- Wrong PIN

`GET_PRIVATE_KEY_FAILED` - error getting a private key from a token

`OPERATION_FAILED`- general operation failure

- `https://127.0.0.1:8440/api/v1/pki/getmessage?nowait` - returns immediately the current status. Possible values [NO_READER, NO_CARD, CARD_ON].

- `https://127.0.0.1:8440/api/v1/pki/getmessage?wait` - waits till the next event occurs.

NOTE: When you plug off the reader with a card on, two messages are displayed: `NO_CARD, NO_READER.` But the first one will be catch with getmessage?wait.

When you plug in a reader with a card on, occures `READER_ON, CARD_ON`. And as a result `READER_ON` will be returned.

- `https://127.0.0.1:8440/api/v1/pki/getreaderon?nowait` - returns immediately with READER_ON if it's attached and NO_READER otherwise.

- `https://127.0.0.1:8440/api/v1/pki/getreaderon?wait` - returns immediately with READER_ON if a reader is attached or waits till it's attached.

- `https://127.0.0.1:8440/api/v1/pki/getcardon?nowait` - returns immediately with NO_READER if a reader isn't attached, NO_CARD if a card isn't inserted or CARD_ON if a card is inserted.

- `https://127.0.0.1:8440/api/v1/pki/getcardon?wait` - returns immediately with NO_READER if a reader isn't attached or wait till the card will be on a reader.

NOTE: It will wait the next tap of a card even if a card is already on a reader.

- `https://127.0.0.1:8440/api/v1/pki/getcardoff?nowait&cardid=<cardid>` - returns immediately with NO_READER if a reader isn't attached, NO_CARD if a card isn't inserted or CARD_ON

if a card is inserted. Use cardid to wait when a specific card is removed.

- `https://127.0.0.1:8440/api/v1/card/getcardoff?wait` - returns immediately with NO_READER if a reader isn't attached. if there is no card on a reader return NO_CARD immediately else waits till the card is removed from the reader

- `https://127.0.0.1:8440/api/abort?cancel-cookie=xxx` - all of the wait methods support cancel-cookie=xxx parameter.

For example, https://127.0.0.1:8440/api/v1/card/getmessage?wait&cancel-cookie=xxx.

And by calling abort with a cancel-cookie, all waiting methods with the same specfied cookie are terminated.

Response format:

```
Response format

{

result: [NO_READER, READER_ON, NO_CARD, CARD_ON],

cardid: <card id>,

readerid: <reader id>

}

cardid is used only with CARD_ON, and NO_CARD result.
```

5.5 Bluetooth Plug-in

To troubleshoot the Bluetooth related issues you can use the following instructions.

The Bluetooth plugin supports the following methods:

1. https://127.0.0.1:8440/api/v1/bluetooth/getdevices

   This GET method returns a JSON array of all discovered bluetooth devices or an error code if Bluetooth is turned off.

   Sample response:

   ```
   {"devices":

   [{"name":"MagicKeyboard","address":"9cd746e1234","type":"peripheral","hash":"9
   b67e2d07088a1f0bd64bde8c44ab7cdc279463bd6d93735ab778afda79d0bde"},

   {"name":"MagicMouse","address":"1abcd22dafae","type":"peripheral","hash":"dbf7
   5830268ab5516a0d658d28105761b6d6ec062a42317a84b3a82e8e4d643f"},

   {"name":"Lex'siPhone","address":"40cd0150cf58","type":"phone","hash":"ac904cc2
   e2626ca27eb7f4100166e0ae07957da89a5a3aa52f0a5d182b6ba42e"}]}
   ```

Fields:

- name - bluetooth device name
- address - bluetooth address of the device
- type - device type [possible types: computer, phone, lan_access, audio, peripheral, imaging, unclassified]\

2. https://127.0.0.1:8440/api/v1/bluetooth/detectdevice

   POST method, used to test the device presence by its address

   The POST body:

   {"address":"[RSA encoded address]"}

   RSA encoded address - a bluetooth address encoded with an RSA public key (from certificate) in a hex-string format.

   If the device is in range, the sevice returns:{"result":"CONNECTED","address":"40cd0150cf58"}

   or if the device is absent or the bluetooth is off on the device:{"result":"DISCONNECTED"}

   Other possible result values for this method:

   - FAILED - general failure
   - DECRYPT_FAILED - decoding failure
   - INVALID_ADDRESS - not a valid value for the address
   - hash - SHA256 hash of the address

   If Bluetooth is off, the call returns the error: `{ "result":"BLUETOOTH_DISABLED"}`

3. https://127.0.0.1:8440/api/v1/bluetooth/getpublickey

   This GET method returns the public certificate in a PEM format. Encode the bluetooth address with the public key in that certificate.

   `{"publicKey":"[PUBLIC_CERT]"}`

   PUBLIC_CERT - the public certificate in a PEM format.

   Currently the following is always same:

   ```
   "-----BEGIN RSA PUBLIC KEY-----\n"
   "MIGHAoGBAKqGJxyB/ZgrTEsfqmMdE4GRwGH+XOioOa0EiQ8+HYcR8Pcg57j1Cc5k\n"
   "D1TrGNKpayWUWW7YEsXvfSpc5a5x9qwsEe06Iak5eP/PcGNLUViLwy2CN9oy5mSM\n"
   "Izpd607GNBUzEwWg0sIpm3FBEvtFFDxBb7PzE9W4hE//t0LQkGcTAgED\n"
   "-----END RSA PUBLIC KEY-----";
   ```