

DnX (Download and Execute)

User Guide

Dec 2019

Revision 1.2

Intel Confidential



By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2019, Intel Corporation. All rights reserved.



Contents

1	Introduction.....	5
	1.1 Terminology.....	5
	1.2 Reference Documents.....	5
2	Download and Execute (DnX)	6
	2.1 Introduction	6
	2.2 Use Cases	7
	2.3 Triggers	7
	2.4 DnX flow.....	8
	2.5 Tools	9
3	High-Level Setup Detail for DnX	10
	3.1 Setup Requirements.....	10
	3.2 Configuration file for partitioning	11
	3.2.1 Important notes:	11
	3.2.2 Example to explain the relationship between LUN size and UFS descriptor parameters:	12
	3.2.3 Attributes supported in configuration file	13
4	Opening DnX capabilities post EOM	15
5	Intel® Platform Flash Tool (PFT) Overview	19
	5.1 Installation Details	19
	5.2 Usage.....	19
	5.2.1 PFT Command Line Tools for DnX	20
	5.2.2 Using GUI interface.....	30
	5.2.3 Common error messages.....	36



Revision History

Revision Number	Description	Revision Date
0.6	<ul style="list-style-type: none">• Overview of DnX feature• Overview of Intel® PFT(Platform Flash Tool)• Detail of DnX support via PFT tool	May 2018
0.8	<ul style="list-style-type: none">• Updated section 4.2.2<ul style="list-style-type: none">○ Get UFS device info○ Reset target platform• Updated section and 4.2.3<ul style="list-style-type: none">○ Added details on read LUNs/Clear RPMB	Sept 2018
0.85	<ul style="list-style-type: none">• Updated PFT “startover” command	December 2018
0.86	<ul style="list-style-type: none">• Added token commands	December 2018
0.87	<ul style="list-style-type: none">• Updated the Read LUN table content	January 2019
0.88	<ul style="list-style-type: none">• Updated PFT “startover” command example.• Added the option to configure to auto size• Clarified “Configuration file for partitioning” section	February 2019
0.89	<ul style="list-style-type: none">• Added chapter 4 Opening DnX capabilities post EOM	March 2019
0.9	<ul style="list-style-type: none">• Added token example	April 2019
0.91	<ul style="list-style-type: none">• Added cflasher command line• Updated details in chapter 5 for the GUI	June 2019
1.0	<ul style="list-style-type: none">• Added a section for attributes configuration (RefClkFreq and bConfigDescrLock)	July 2019
1.1	<ul style="list-style-type: none">• Added HPB support	Oct 2019

§ §



1 Introduction

The purpose of the document is to provide guidance on the Download and Execute (DnX) feature, usage this feature and how it gets enabled using Intel® CSE components as well as Intel® Platform Flash Tool (PFT).

1.1 Terminology

Table 1: Terminology

Acronym or Term	Definition
Intel® CSE	Intel® Converged Security Engine
DnX	Download and Execute
Intel® FIT	Intel® Flash Image Tool
FW	Firmware
Intel® PFT	Intel® Platform Flash Tool

1.2 Reference Documents

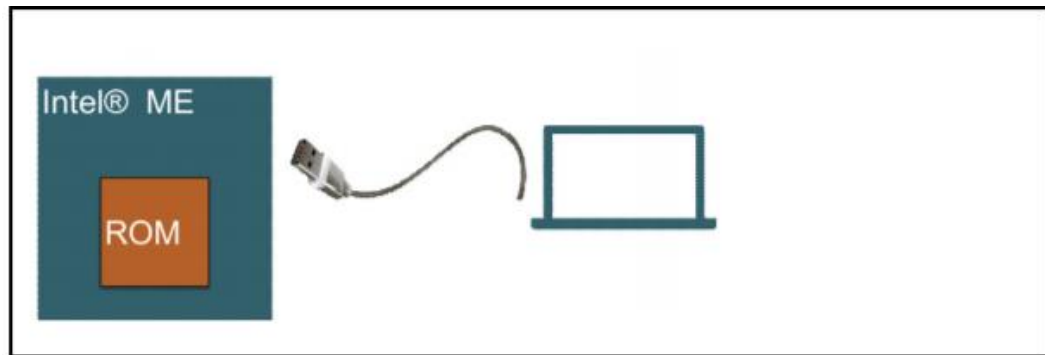
Table 2: Reference Documents

Document	Document No. / Location
Lakefield Intel® CSE FW 13.30 PRD	CDI# 576361

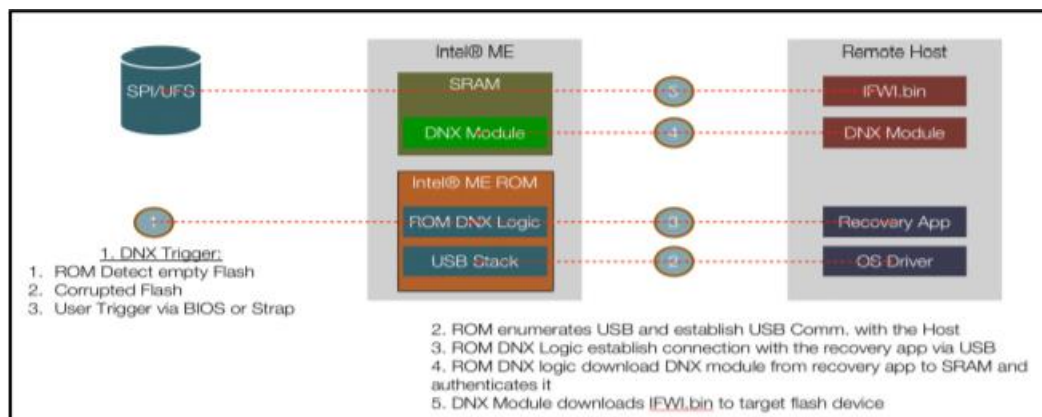
2 Download and Execute (DnX)

2.1 Introduction

DnX is Intel's proprietary solution to download FW module to a target machine from a host machine by means of USB cable and execute it. DnX flows are executed over fixed USB 2 port. On UFS platforms, this capability allows to access boot media for IFWI write, read content of UFS partitions as well as signed Token injection for debug unlock after the platform have completed manufacturing.



There are many advantages in using DnX, one of which is the capability to flash IFWI without opening the chassis to physically reach the flash device. This is especially important in the manufacturing line once the chassis is closed, in debug labs and in post manufacturing scenarios. It can be used to establish the internal partitions on UFS flash device and flash the IFWI on a blank UFS device pre manufacturing. Other methods such as FWUpdate and Capsule flow can be used on subsequent upgrades/downgrades if the platform can boot to an operating system. DnX is a capability in Intel® CSE ROM, where during the boot ROM can configure the USB port #0 on the PCH to connect to a remote computer to download DnX module which is signed by Intel. This module initiates rest of the Intel® CSE and setup an environment to accept IFWI binary from a remote computer. The flow is explained below:



2.2 Use Cases

Below use-cases are supported on Lake field platform.

Note: Currently, DnX is POR for UFS boot media and not POR for SPI boot media

Scenario	Use Case
Manufacturing	<ul style="list-style-type: none"> • UFS Device Configuration(set size of LUNs and attributes) • Full IFWI.bin programming on to UFS device boot partition • Clear Data(CSE and BIOS) in RPMB • Read Contents of UFS partitions
Debug	<ul style="list-style-type: none"> • Create/Write/Read/Erased signed debug tokens into the platform

2.3 Triggers

Following methods can be used to trigger DnX on the platform

Method	Detail
Pin Strap	GPIO209_FORCE_FW_RELOAD can be used to trigger DnX on the platform. For



Method	Detail
	details refer to “Lakefield SoC External Design Specification (EDS) Volume 1 of 2 - Doc# 574211”
Intel® CSE or BIOS Error Handling Flow	If Intel® CSE or BIOS reach a critical error which prevent platform from booting, it can be programmed to enter DnX flow. (e.g Failure to authenticate BIOS signature, CSE detect FW corruption, etc.)
Empty Flash Device	When CSE ROM detects an empty flash device on the platform, it will enter DnX mode
BIOS HECI Call	BIOS can make a HECI call to Intel® CSE during boot to enter DnX after the reset

2.4

DnX flow

1. DNX Trigger:

- a. ROM Detect empty Flash
- b. Corrupted Flash
- c. User Trigger via BIOS or Strap

2. ROM enumerates USB and establish USB Comm. with the Host

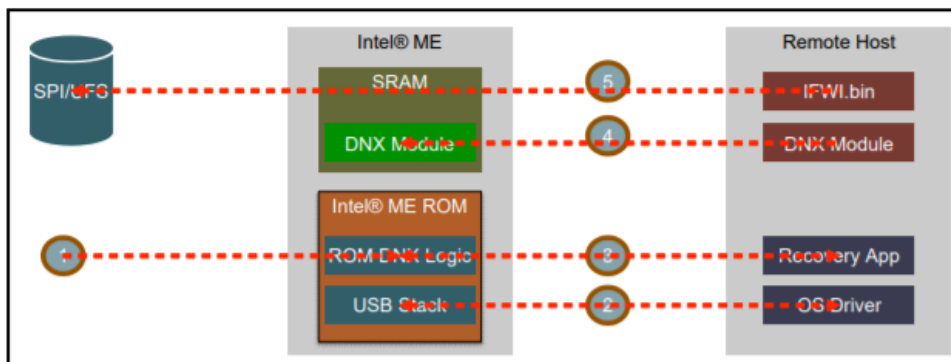
3. ROM DNX Logic establish connection with the recovery app via USB

4. ROM DNX logic download DNX module from recovery app to SRAM and authenticates it

5. DNX Module performs DnX operation requested by user (e.g. configures the UFS, reads LUN content, downloads IFWI.bin to target flash)



Intel® DNX Flow



2.5 Tools

Following tools are applicable for DnX:

- **Intel® PFT** (Platform Flash Tool) – Intel implementation of DnX tool running on remote host computer. DnX module, config.xml and IFWI.bin are inserted to the target machine via this tool. Will be included in the Intel® CSE Kit for Lakefield platform.
- **DnX Module** - binary file signed by Intel. This file has the DnX logic Intel® CSE ROM will run. Will be included in the Intel® CSE kit for Lake field Platform.
- **Intel® FIT** – can be used to create DnX based IFWI image. For more detail on how to create IFWI image for DnX, please refer to Intel® Bring up Guide in the Intel® CSE kit for Lakefield Platform.
- **CFGFile.xml** – configuration file used to create partitions (LUNs) on UFS device. This file is used as input by Intel® PFT tool and will be included in the Intel® CSE Kit for Lakefield platform. OEMs are expected to update this file according to LUN partition numbers and size they are going to use for UFS device.
- **Intel® FPT** - can be used to configure DnX fuse and close manufacturing on the platform. Will be included in the Intel® CSE kit for Lake field Platform.

3 High-Level Setup Detail for DnX

3.1 Setup Requirements

In order to complete DnX flow, the following setup is required

DnX Test Setup

Target Device

- Enters DnX mode based on trigger

Management Console

- Platform Flash Tool (PFT)
- DnX module
- IFWI image



Requirements:

Requirement	Usage
Management Console / Remote Host	A host that can be used to execute the DnX flows
H/W Connection	USB cable connection between the Remote Host to the system under test
Intel® Platform Flash Tool (PFT)	Tool supporting DnX flows running on the Remote Host
DnX module binary	Provided in the Intel® CSE FW kit. This binary is provided as an input to the Intel® PFT.
DnX based IFWI image	IFWI image to be flashed on the target system (Can be created by Intel® FIT tool provided in the Intel® CSE)
CFGFILE.XML	Required if UFS device needs to be partitioned. Provided in the Intel® CSE FW kit. This file is provided as an input to the Intel® PFT



3.2 Configuration file for partitioning

DnX can be used to set the UFS LUNs descriptor using configuration file.

Each LUN descriptor appears in the XML with the following lines:

```
<lun idx="1">
    <enable>true</enable>
    <boot-lun-id>0x1</boot-lun-id>
    <write-protect>0x0</write-protect>
    <mem-type>0x0</mem-type>
    <alloc-units>0x8000000</alloc-units>
    <data-reliability>0x0</data-reliability>
    <logical-block-size>0xc</logical-block-size>
    <provisioning-type>0x0</provisioning-type>
    <ctx-caps>0x0</ctx-caps>
</lun>
```

Configuration file needs to be updated per requirement before creating partition.

For basic usage, it is recommended to only change the: <enable>, <boot-lun-id> and <alloc-units> lines.

For UFS part supporting HPB, new section called “ufs_hpb” was added to the configuration file. DnX FW will consume this part of the file only when device type “ufs_hpb” is specified in the “configpart” command.

3.2.1 Important notes

- If a LUN does not appear in the XML, it will be set as disabled.
- <alloc-units> is in big-endian format (as UFS part expects it).
- Enabling LUN with size of 0 is valid by the DnX SW tool. If this size is not supported by the UFS device, it will return an error to the DnX SW tool which will then prompt the error to the user.



- DnX SW supports allocating all the unused storage to the chosen LUN:

<alloc-units>**auto**</alloc-units>

OR:

<alloc-units>**-1**</alloc-units>

Only one LUN can be configured with 'auto' (-1), otherwise DnX SW tool will raise an exception.

3.2.2 Example to explain the relationship between LUN size and UFS descriptor parameters

If UFS spec states the following:

- dNumAllocUnits (which is the same as <alloc-units>) has 4 Bytes size
- bAllocationUnitSize has 1 Bytes size
- dSegmentSize has 4 Bytes size
- The Capacity Adjustment Factor value for Normal memory type is one
- $LUN\ Size = (bAllocationUnitSize * dSegmentSize * dNumAllocUnits \times 512) / CapacityAdjFactor$

And the values that had been read from the UFS using "getcardinfo" command (see "Get UFS device detailed info" section below) are:

- bAllocationUnitSize = 0x01
- dSegmentSize = 0x00200000

Then:

First let's swap bAllocationUnitSize and dSegmentSize to little endian:

Byte Swap bAllocationUnitSize from Big Endian into Little Endian: 0x01 = 1

Byte Swap dSegmentSize from Big Endian into Little Endian: 0x 00 00 20 00 = 2^{13}

Example #1: <alloc-units>**0x1000000**</alloc-units>



Complete to 4 Bytes: 0x 01 00 00 00

Byte Swap from Big Endian into Little Endian: 0x 00 00 00 01

LUN Size = $(1 * 2^{13} * 1 \times 512) / 1 = 4\text{MB}$

Example #2: <alloc-units>**0x2000000**</alloc-units>

Complete to 4 Bytes: 0x 02 00 00 00

Byte Swap from Big Endian into Little Endian: 0x 00 00 00 02

LUN Size = $(2 * 2^{13} * 1 \times 512) / 1 = 8\text{MB}$

Example #3: <alloc-units>**0x8000000**</alloc-units>

Complete to 4 Bytes: 0x 08 00 00 00

Byte Swap from Big Endian into Little Endian: 0x 00 00 00 08

LUN Size = $(8 * 2^{13} * 1 \times 512) / 1 = 32\text{MB}$

Example #4: <alloc-units>**0x10000**</alloc-units>

Complete to 4 Bytes: 0x 00 01 00 00

Byte Swap from Big Endian into Little Endian: 0x 00 00 01 00

LUN Size = $(2^8 * 2^{13} * 1 \times 512) / 1 = 1\text{GB}$

Example #5: <alloc-units>**0x100**</alloc-units>

Complete to 4 Bytes: 0x 00 00 01 00

Byte Swap from Big Endian into Little Endian: 0x 00 01 00 00

LUN Size = $(2^{16} * 2^{13} * 1 \times 512) / 1 = 256\text{GB}$

3.2.3 Attributes supported in configuration file

Intel® CSE DnX supports setting following attributes:

- bRefClkFreq



- bConfigDescrLock

Description and default values of those attributes can be found in UFS spec.

Those attributes can be set to the desired value thru DnX Configuration file.

3.2.3.1 bRefClkFreq

UFS out of vendor has to be configured with bRefClkFreq = 19.2MHz so that DnX mode can be triggered automatically because this is the clock driven by PCH. In cases where UFS out of vendor has bRefClkFreq configured to value other than 19.2MHz, DnX has to be triggered manually using “Pin Strap” method only.

3.2.3.2 bConfigDescrLock

When partitioning with “config-descr-lock” value set to 0x0, UFS Configuration Descriptor will not be locked. But when running configuration command with “config-descr-lock” value set to 0x1, Configuration Descriptor will get locked and re-partitioning will no longer be possible. This attribute can be written only one time, the value is kept after power cycle or any type of reset event.

3.2.3.3 Example of setting the attributes value in configuration file:

```
<attributes configure = "true">  
  <ref-clock-freq configure="true">0x0</ref-clock-freq>  
  <config-descr-lock configure="true">  
    <value>0x0</value>  
  </config-descr-lock>-->  
  
</attributes>
```



4 Opening DnX capabilities post EOM

Table below lists all DnX capabilities and how they are affected by OEM:

Operation	SOC_ConfigLock is not set	SOC_Config Lock is set	Can be re- enabled through DnX token
START-OVER	Allowed	Allowed	N/A
ID Device (PING)	Allowed	Allowed	N/A
Download Recovery Module	Allowed	Allowed	N/A
Download OEM Key Manifest	Allowed	Allowed	N/A
Get NV Store Info	Allowed	Prohibited	Yes
Configure Device	Allowed	Prohibited	Yes
Clear Data	Allowed	Prohibited	Yes
Provision FW Image	Allowed	Prohibited	Yes
Set Capabilities	Allowed	Allowed	N/A
Get Token Part ID	Allowed	Allowed	N/A
Read Token	Allowed	Allowed	N/A
Write Token	Allowed	Allowed	N/A
Erase Token	Allowed	Allowed	N/A
Read Boot Media Content	Allowed	Prohibited	Yes

Some capabilities, like secure token handling operations, are always available.

Note: in order to successfully write/read/erase secure token, “logical” sub-partition called UTOK must be present on the UFS. When flashing IFWI on UFS with new (or changed) configuration, Intel® CSE reset is required for UTOK (and other sub-partitions) to get created by Intel® CSE FW as a part of file system deployment.



After first boot, UTOK is in Temporary Data partition/RPMB and token operations can be executed.

However, DnX capabilities allowing access the boot media are prohibited once platform went through EOM (End of Manufacturing). In order to enable them, OEM authorization is required.

This authorization is provided by generating an OEM unlock token with “DnX capabilities” knob enabled. Value set in this knob will specify which of the DnX capabilities will be allowed for the specific session where token is valid.

Flow to open prohibited post EOM DnX capabilities:

1. Use PFT to generate new OEM unlock token with active “DnX Capabilities” knob. Set value of this knob according to the desired DnX capability (see Figure 4.1 and 4.2 for example). To see more details on how to generate a token refer to “LKF Secure Tokens Guide”
2. Sign OEM unlock token using PFT. To see more details on how to sign a token refer to “LKF Signing and Manifesting Guide”.
3. Prepare signed OEM KM containing:
 - a. Public key hash of private key used for signing OEM unlock token
 - b. Public key hash of private key used for signing DnX Image
4. Use PFT to run DnX command to download OEM KM to the SRAM (need to provide OEM KM and DnX Module as input). See command example in “Open DnX capabilities post EOM” section below.
5. Use PFT to run DnX command to set capabilities defined in OEM unlock token (need to provide OEM unlock token and DnX Module as input). See command example in “Open DnX capabilities post EOM” section below.

Example of how to enable DnX capabilities knob inside OEM unlock token using PFT GUI:

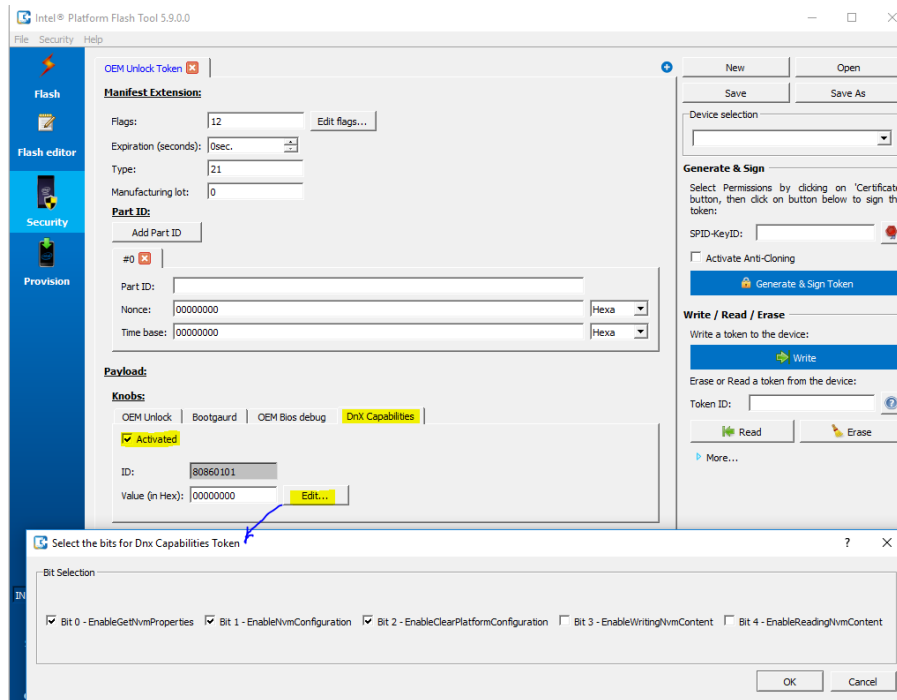


Figure 4.1

Example of how to enable DnX capabilities knob inside OEM unlock token using xml inside PFT installation folder (oem_unlock_token_template_lakefield.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<tokens version="1" format="1" view_filter="simple">
  <token name="OEM Unlock Token" platform="Lakefield" token_id="3">
    <!--Set the desired expiration time for token to be valid-->
    <manifest_extension type="21" length="0" version="1" number_ids="1" payload_version="1" flags="0" expiration_seconds="3600" manufacturing_lot="0">
      <!-- Enter part_id, nonce and time base of your device -->
      <part_id nonce="00000000" part_id="0x" time_base="00000000"/>
    </manifest_extension>
    <payload nb_knobs="4">
      <knob name="OEM Unlock" id="0x80860002" data="0x00000001" activated="1"/>
      <knob name="Bootguard" id="0x80860030" data="0x00000002" activated="1"/>
      <knob name="OEM Bios debug" id="0x80860051" data="0x00000001" activated="1"/>
      <!--"activated" points to which knob is activated-->
      <!--"data" depends on the list below. The data is in hex-->
      <Bitmap options:>
        <EnableGetNvmProperties - bit 0>
        <EnableNvmConfiguration - bit 1>
        <EnableClearPlatformConfiguration - bit 2>
        <EnableWritingNvmContent - bit 3>
        <EnableReadingNvmContent - bit 4-->
      </Bitmap options>
      <knob name="DnX Capabilities" id="0x80860101" data="0x00000001" activated="1"/>
    </payload>
  </token>
</tokens>
```

Figure 4.2



In this example data="0x0000001f" means that bits[4..0] = '0b11111', hence **all** DnX capabilities are open for the DnX session where above token is valid.

In order to provide debug support for OEM platforms post EOM, Intel has the capability to open DnX for specific platform.



5 Intel® Platform Flash Tool (PFT) Overview

Intel® Platform Flash Tool supports GUI (Graphic User Interface) as well as CLI (Command Line Interface) and runs on the Remote Host.

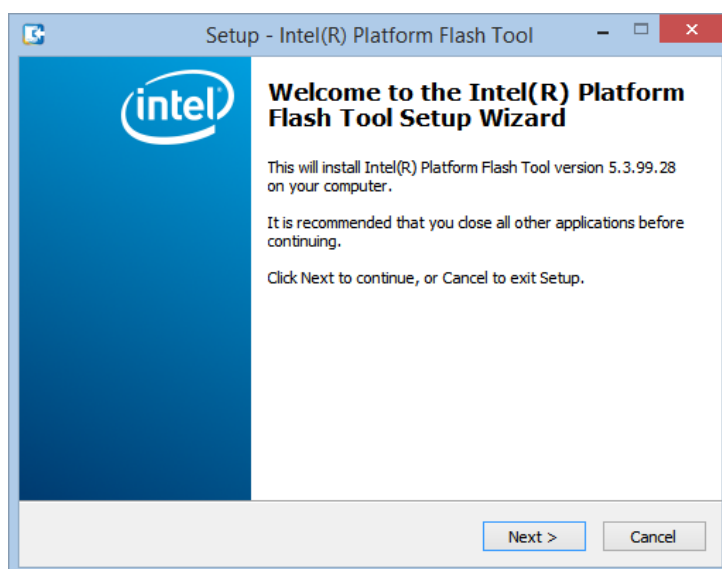
This tool supports DnX flows and consumes DnX related input files like: DnX module, configuration file as well as DnX IFWI planned to be flashed on the target system.

Please install this tool on Host system before executing DnX flows.

5.1 Installation Details

PFT tool is available within Intel® CSE FW Kit->Tools->DnX Tools. Run the installation package. Setup wizard will start. Click “Next” to complete the installation.

This installation process installs Tools as well as necessary USB drivers along with it as well.



5.2 Usage

Once PFT tool is installed on the Remote Host:

- Make sure the target system is connected to the Remote Host using USB cable.
- Make sure all input files required for DnX operation (e.g. DnX module, configuration file for UFS partition creation, DnX IFWI) are available on the Remote Host.



- When using PFT GUI, up to 8 targets can be programmed in parallel, no such limitation when using PFT command line. There are not limitation on parallelism from DnX side.

5.2.1 PFT Command Line Tools for DnX

- **DnX Firmware Downloader**

This command line tool provides means to interact with Intel® CSE firmware and perform different DnX operation (configuring partitions, flashing, getting information, etc.)

This tool supports serial number argument, however does not provide USB port hence less convenient for running DnX commands on multiple targets in parallel.

- **cflasher**

Provides a wrapper for *dnxFwDownloader*, assigning USB port to each target based on its physical location on the USB hub

Multiple instances of the *cflasher* tool can be launched in parallel. In this case the user has to provide the target serial number with the option `--dnx-sn`.

The `-f` option is mandatory for *cflasher*.

Note: in order to run any of those tools, target machine has to be already in DnX mode (e.g jumper, virgin UFS)

5.2.1.1 DnX Firmware Downloader

Usage:

dnxFwDownloader --command <command> <command-options>

Help Menu

dnxFwDownloader.exe --help command lists available options/commands supported with this embedded tool.

For all the commands requiring device type as argument in the command line, new device type “ufs_hpb” is supported in addition to “ufs”.

5.2.1.1.1 Get UFS device general info

In order to get UFS related information such as OEM PLAT ID (from IFPs), Platform Unique ID, DnX Trigger, Image Error Values, ***'iddevice'*** command shall be used.

**Sample:**

```
dnxFwDownloader.exe --command iddevice
```

5.2.1.1.2 Get UFS device detailed info

In order to get UFS details like Card Identification, Card Specific Data, Operations Conditions Register (contains voltage related info of the UFS), '**getcardinfo**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command getcardinfo --fw_dnx DNXP_0x1.bin --device ufs --idx 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--device	Boot device type (ufs)
--idx	device index[Optional]

5.2.1.1.3 UFS device Configuration/Partition Creation

In order to configure UFS partitions, '**configpart**' command shall be used.

Note: Please update cfgpart.xml according to # of LUN partitions and size expected to be used for UFS device before using this command. For basic usage, it is recommended to only change the enable, boot-lun-id and alloc-units lines.

Sample:

```
dnxFwDownloader.exe --command configpart --fw_dnx DNXP_0x1.bin --path cfgpart.xml --device ufs --idx 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--path	path to configuration file (XML format) for LUN creation



--device	Boot device type (ufs)
--idx	device index[Optional]

5.2.1.1.4 Flash IFWI via DnX

In order to flash IFWI image, '**downloadfwos**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command downloadfwos --fw_dnx DNXP_0x1.bin --fw_image  
IFWI.bin --flags 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--fw_image	path to the firmware image
--flags	firmware download command flags[Optional]

5.2.1.1.5 Reset Target Platform

In order to reset the platform, '**startover**' command shall be used.

Sample:

```
dnxFwdownloader.exe --command startover --flags 9
```

Where:

Option	Description
--flags	Firmware reset command flags. In the command line appear in decimal display. Comprises of following info (in binary): Bit [1:0]: RESET_TYPE* <ul style="list-style-type: none">00: Reset DnX protocol (no Intel® CSE /device



	<p>reset) by cancelling currently active command (if any) and wait for the next command</p> <ul style="list-style-type: none"> • 01: Global reset • 10: Not supported, returns error • 11: Not supported, returns error <p>Bit [3:2]: POST_RESET_STEPS</p> <ul style="list-style-type: none"> • 00: After reset, take normal boot path (including honoring the DnX triggers etc.) • 01: After reset, enter OS DNX flow • 10: After reset, ignore optional DnX triggers such as HW strap etc. and perform a full host boot • 11: Reserved
--	---

5.2.1.1.6 Read LUN content

In order to read content of specific LUN, '**readbootmedia**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command readbootmedia --fw_dnx DNXP_0x1.bin --path dumpLUN0.bin --device ufs --idx 0 --start 0 --blocks 4096 --part 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--path	path to output file to dump the content of the LUN
--device	Boot device type (ufs)



--idx	device index[Optional]
--blocks	Number of blocks to read where each block size: 1 block = 1kByte E.g. to read 4MB LUN: 4MB = 4096kB (in binary) = 4096 block
--part	Partition number: 16 – LUN0 (User partition) 17 (or 0) – LUN1 (Boot partition) 18 – LUN2 19 – LUN3 20 – LUN4 21 – LUN5 22 – LUN6 (Temp data partition) 48 – RPMB partition

Returned data is in the 'raw' as read from the media and is not processed at all by DnX module (i.e. no decryption etc. is performed, rather all data is returned as stored on the media)

Make sure that output file location can be accessed for write, otherwise operation will fail.

Make sure to set correct number of blocks to read based on the partition size

5.2.1.1.7 Clear RPMB

In order to clear RPMB, '**clearrpmb**' command shall be used.

Sample:



*dnxFwDownloader.exe --command **clearrpmb** --fw_dnx **DNXP_0x1.bin** --idx 0 --device **ufs***

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--device	Boot device type (ufs)
--idx	device index[Optional]

5.2.1.1.8 Read Token

In order to read token, '**readtoken**' command shall be used.

Sample:

*dnxFwDownloader.exe --command **readtoken** --fw_dnx **DNXP_0x1.bin** --path **read_token.bin** --slot 0*

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--path	path to output file to dump the content of the token
--slot	Slot Index of the token

5.2.1.1.9 Write Token

In order to write token, '**writetoken**' command shall be used.

Sample:

*dnxFwDownloader.exe --command **writetoken** --fw_dnx **DNXP_0x1.bin** --token **token_to_write.bin** --slot 0*

Where:

Option	Description
--fw_dnx	path to the DnX module binary



--token	path to the token
--slot	Slot Index of the token

5.2.1.1.10 Erase Token

In order to erase token, '**erasetoken**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command erasetoken --fw_dnx DNXP_0x1.bin --slot 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--slot	Slot Index of the token

5.2.1.1.11 Get Token Part ID

In order to get part ID specific to this token, '**gettokenpid**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command gettokenpid --fw_dnx DNXP_0x1.bin --flags 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--flags	Slot number for anti-replay protection of corresponding token: <ul style="list-style-type: none">• 0: No AR protection needed. Nonce is stored in the temp storage in SRAM• 1: Nonce generated is stored in first Nonce slot



5.2.1.1.12 Open DnX capabilities post EOM

In order to download OEM KM as part of the flow to open prohibited post EOM DnX capabilities, '**downloadoemkeymanifest**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command downloadoemkeymanifest --key OEM_KM.bin --fw_dnx DNXP_0x1.bin
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--key	Path to the OEM Key Manifest binary

In order to set DnX capabilities enabled in OEM unlock token as part of the flow to open prohibited post EOM DnX capabilities, '**setcapabilities**' command shall be used.

Sample:

```
dnxFwDownloader.exe --command setcapabilities --capabilities OEM_UnlockToken.tok --fw_dnx DNXP_0x1.bin
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary
--capabilities	<p>Path to the OEM Unlock Token with DnX capabilities knob enabled and set to the desired value:</p> <p>DnX capabilities knob options:</p> <ul style="list-style-type: none"> Bit 0 - EnableGetNvmProperties Bit 1 - EnableNvmConfiguration Bit 2 - EnableClearPlatformConfiguration Bit 3 - EnableWritingNvmContent Bit 4 - EnableReadingNvmContent



5.2.1.2 Cflasher

This tool takes json file as input. Json file can contain one or more DnX commands that can be automatically run one after another (according to their order in json file) or by separate *cflasher commands*.

- Json file for example is provided in LKF release kit.
- All the input files (like DnX module, configuration file, DnX IFWI etc...) should be in the same folder as json.
- “commands” section of the json file will define the order of executed commands (as order in which they appear inside the file), while “configurations” section will defined their names.
- To run **all** the commands defined in json:

```
cflasher.exe -f LKF_IFWI.json
```

- To run **only** “downloadfwos” command from json:

```
cflasher.exe -f LKF_IFWI.json -c “ifwi_flash”
```

(“ifwi_flash” in this case is the configuration name chosen for “downloadfwos” command)

- To run **all** the commands on **specific target** with known serial number:

```
cflasher.exe -f LKF_IFWI.json --dnx-sn “8210995e6a8ce5edaf36fa34d4069078”
```

(serial number in this case is the iddevice)




```

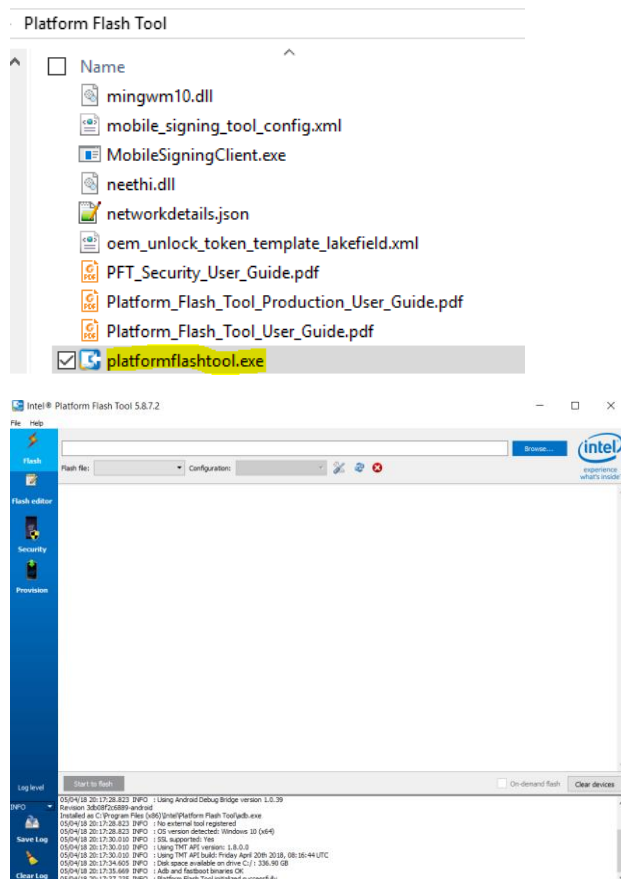
"flash" : {
  "commands" : [
    {
      "command" : "iddevice",
      "description" : "ID Device",
      "flags" : "${flags_iddevice}",
      "mandatory" : true,
      "restrict" : [
        "iddevice",
        "default"
      ],
      "retry" : 2,
      "timeout" : 60000,
      "tool" : "dnxFwDownloader"
    },
    {
      "command" : "configpart",
      "description" : "Configuration",
      "fw_dnx" : "${fw_dnx_file}",
      "path" : "${config_file}",
      "device" : "${device}",
      "idx" : "${idx}",
      "mandatory" : true,
      "restrict" : [
        "configure",
        "default"
      ],
      "retry" : 2,
      "timeout" : 60000,
      "tool" : "dnxFwDownloader"
    },
    {
      "command" : "downloadfwos",
      "description" : "Flashing IFWI",
      "flags" : "${flags_downloadfwos}",
      "fw_dnx" : "${fw_dnx_file}",
      "fw_image" : "${fw_image_file}",
      "mandatory" : true,
      "restrict" : [
        "ifwi_flash",
        "default"
      ],
      "retry" : 2,
      "timeout" : 60000,
      "tool" : "dnxFwDownloader"
    },
    {
      "command" : "startover",
      "description" : "Start Over",
      "flags" : "${flags_startover}",
      "mandatory" : true,
      "restrict" : [
        "startover",
        "default"
      ],
      "retry" : 2,
      "timeout" : 60000,
      "tool" : "dnxFwDownloader"
    }
  ],
  "configurations" : {
    "default" : {
      "brief" : "Default configuration",
      "default" : true,
      "description" : "Default configuration",
      "documentation" : "",
      "groupsState" : {
      },
      "hiddenToEndUser" : false,
      "name" : "Default",
      "parameters" : {
      },
      "startState" : "dnx_fw",
      "warning" : ""
    },
    "iddevice" : {
      "brief" : "ID Device",
      "default" : false,
      "description" : "ID Device",
      "documentation" : "",
      "groupsState" : {
      },
      "hiddenToEndUser" : false,
      "name" : "ID Device",
      "parameters" : {
      },
      "startState" : "dnx_fw",
      "warning" : ""
    },
    "configure" : {
      "brief" : "Configuration device",
      "default" : false,
      "description" : "Configuration device",
      "documentation" : "",
      "groupsState" : {
      },
      "hiddenToEndUser" : false,
      "name" : "Configure",
      "parameters" : {
      },
      "startState" : "dnx_fw",
      "warning" : ""
    },
    "ifwi_flash" : {
      "brief" : "IFWI Flash",
      "default" : false,
      "description" : "IFWI Flash",
      "documentation" : "",
      "groupsState" : {
      },
      "hiddenToEndUser" : false,
      "name" : "IFWI Flash",
      "parameters" : {
      }
    }
  }
}

```



5.2.2 Using GUI interface

To launch GUI interface, click on the Desktop icon  which launches GUI interface or open it from Intel® PFT installation folder.



5.2.2.1 Executing json file containing DnX commands

In order to execute json file with DnX commands:

1. Choose “Flash” tab on the left panel of PFT
2. Use “Browse” button to load desired json file
3. Choose .json or .xml or local archive file which has set of attributes defined to perform DnX use cases. Sample json file is provided with Intel® PFT Tool folder available within Intel® CSE FW Kit.
4. Click on “Start Flash” Tab.

Note: User is expected to update this sample file with appropriate naming and path details of configuration xml file, DnX module and DNX IFWI binary.

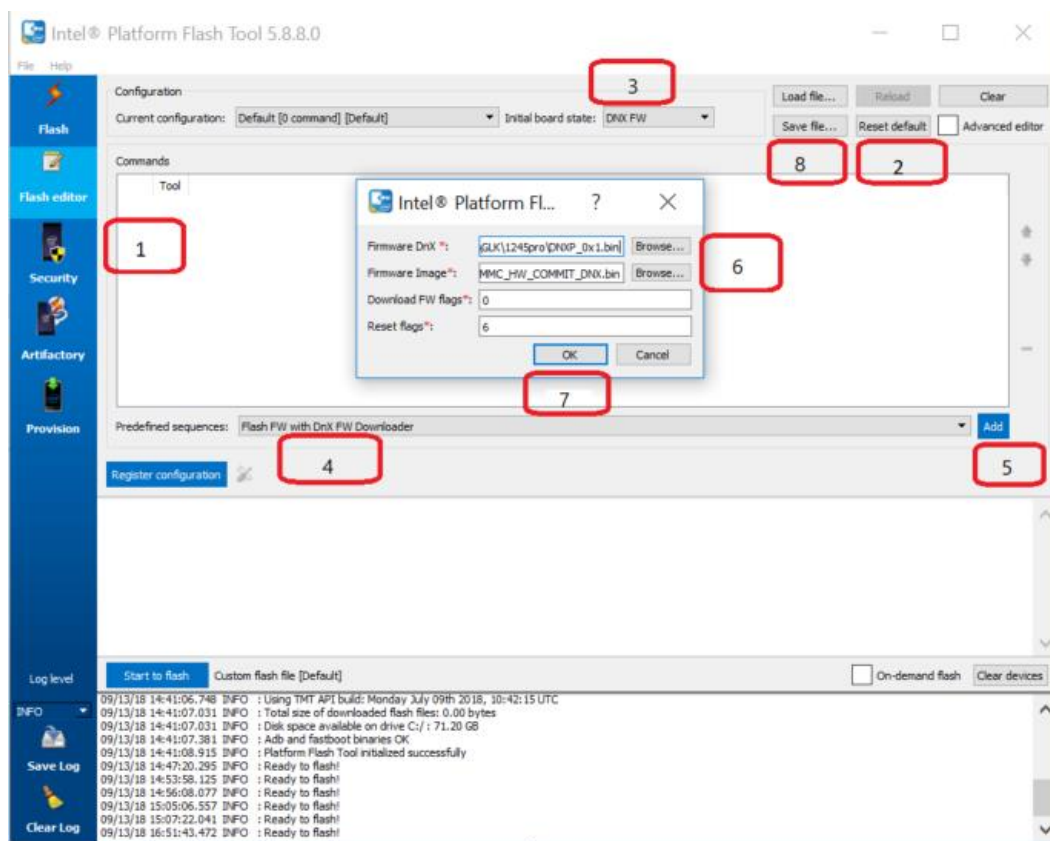


5.2.2.1.1 Creating json file for flashing Intel® CSE IFWI thru DnX

In order to create json file supporting IFWI flash command via DnX from scratch, follow those steps:

Open PFT GUI

1. Choose **"Flash editor"** tab on the left panel of PFT
2. Click **"Reset Default"** button if the flash editor contains any data
3. Set **"Initial board state"** to **"DNX FW"**
4. Set **"Predefined sequence"** to **"Flash FW with DnX FW Downloader"** and click **"Add"**
5. After clicking **"Add"**, a new Window will open:
 - a. In the **"Firmware DnX"** field, select the path to the DnX module. The DnX module is part of the IFWI kit, provided by Intel (Usually named "dnxp_0x1.bin")
 - b. In the **"Firmware Image"** field, select the path to the IFWI image you would like to flash
 - c. Set Reset flags according to desired type of reset (see **"Reset Target Platform"** section of this document for reference)
6. Click **"Ok"** and then **"Save file"**
7. After saving the file, you can load it in later time to flash the image or, you can click the **"Start to flash"** button to flash the image right away



5.2.2.1.2 Executing different DnX operations defined in one json file

It is possible to support number of DnX operations in one json file, each operation will appear as “command” when reviewing json file in text editor (see example of such json below). Operations can be run in GUI one-by-one by changing “**Configuration**” option for each action in the drop down menu. Also can define one configuration that will run number of operations one after another.

In the example below there are three DnX operations defined in one json: “configpart” (configuration), “dnxFwDownloader” (IFWI write) and “startover” (reset). All those operations will run under configuration name “default”, while “configpart” will also run under configuration “configure”.

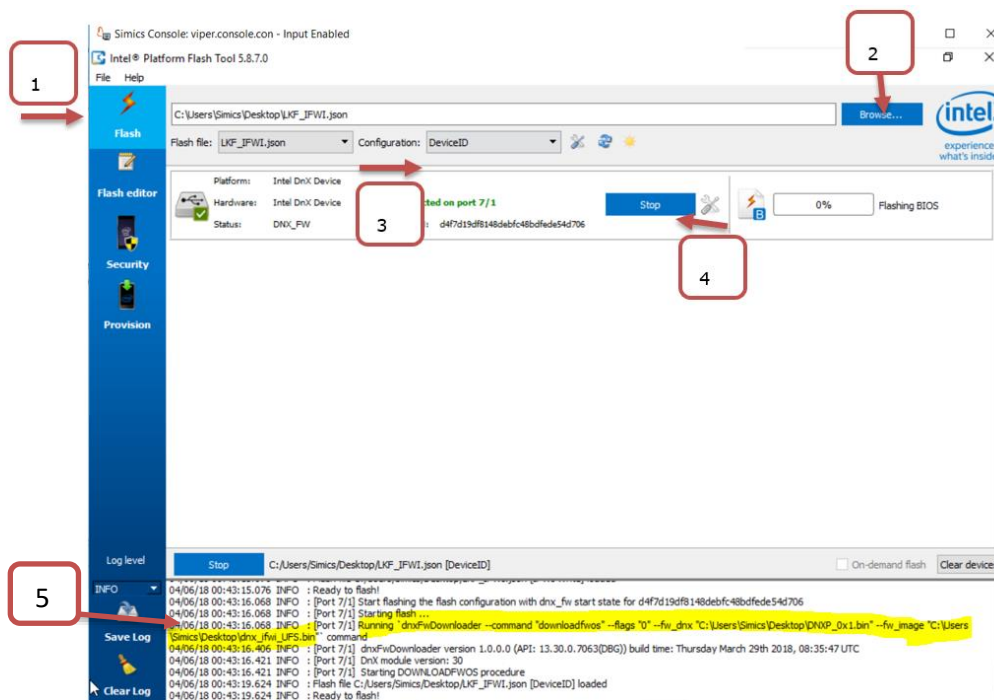


```

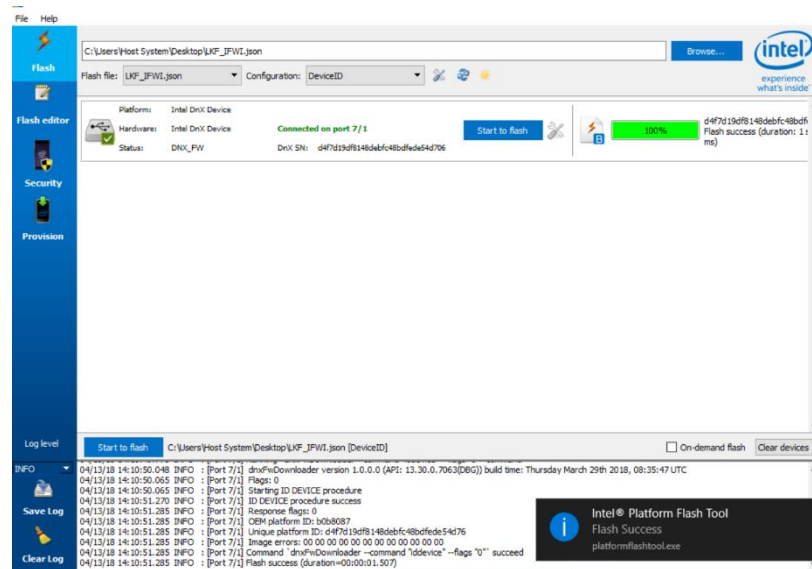
"commands" : [
{
  "command" : "configpart",
  "description" : "Configuration",
  "fw_dnx" : "${fw_dnx_file}",
  "path" : "${config_file}",
  "device" : "${device}",
  "idx" : "${idx}",
  "mandatory" : true,
  "restrict" : [
    "configure",
    "default"
  ],
  "retry" : 2,
  "timeout" : 60000,
  "tool" : "dnxFwDownloader"
},
{
  "command" : "downloadfwos",
  "description" : "Flashing IFWI",
  "flags" : "${flags_downloadfwos}",
  "fw_dnx" : "${fw_dnx_file}",
  "fw_image" : "${fw_image_file}",
  "mandatory" : true,
  "restrict" : [
    "ifwi_flash",
    "default"
  ],
  "retry" : 2,
  "timeout" : 60000,
  "tool" : "dnxFwDownloader"
},
{
  "command" : "startover",
  "description" : "Start Over",
  "flags" : "${flags_downloadfwos}",
  "mandatory" : true,
  "restrict" : [
    "startover",
    "default"
  ],
},

```

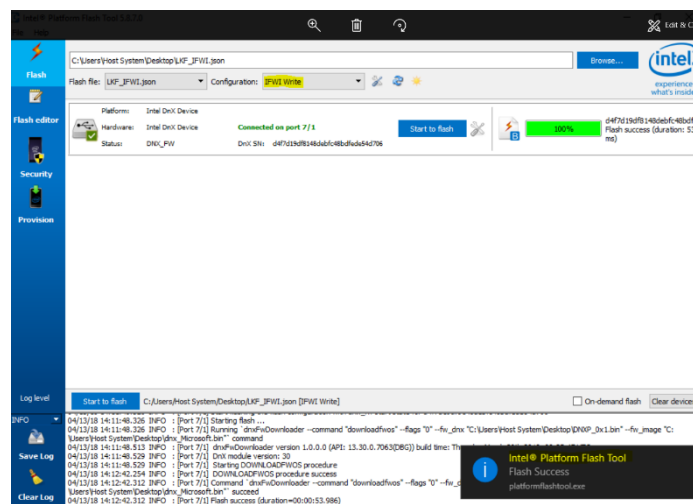
Below example demonstrates json with two DnX commands: Device ID and IFWI write



- Select **“Flash”** tab on the eft panel of the GUI.
- Select the json file using the **“Browse”** button. The json file validity is then checked and the flash operation can be started only if the selected flash file is valid. The details of the loaded flash file are printed in the log area in the DEBUG log level.
- Once LKF_IFWI.json file is selected and it is loaded successfully, under **“Configuration”** drop down menu, there are 2 options – **“Device ID”** and **“IFWI write”**.
- Select **“Device ID”** option.
- Once configuration option is selected, click on **“Start Flash”** Tab.
- On success, Host and Target communication is established and deviceid is presented.



- Now from configuration drop down menu, select “**IFWI Write**” option.
- Click on “Start Flash” Tab.
- This will initiate the IFWI flash process via DnX. After successfully IFWI flash, reset the target platform.





5.2.3 Common error messages

Error Code	Detail
0x30003 - Device not found	<p>If the DNX device isn't available in the device manager: Make sure you are following the correct steps to enter DNX mode, power cycle your platform since DNX has a five minutes timeout (you might need to disconnect the USB cable when power cycling), check your USB cable is ok and is connected to the correct ports, check if any reworks are needed on your platform.</p> <p>If the DNX device is available in the device manager, then the API/Tool version and the DNX driver version used are incompatible.</p>
0x20000007 - Internal system error	<p>This error is caused when using a non-matching DNX SW and DNX driver. Make sure you use the latest DNX driver and DNX SW.</p>
0x30000 - basic_ios::clear	<p>This means there's either a syntax error or the files given to the command don't exist or are incorrect.</p>
Error code: 0x80000000 - Media not present	<p>Boot device chosen by the hard strap is different from the device issued in the command. DNX uses the boot device chosen by the hard strap, make sure strap and command match.</p>
0x80000008 - Invalid public key	<p>This means your DNX module (DNXP_0x1.bin) is signed with the wrong key. Usually this happens when trying to load the debug signed DNX module on production platforms. You need to use a production signed DNX module on</p>



Error Code	Detail
	production platforms.
Error code: 0x80000021 - Invalid partition header	This means that the DNX IFWI you are trying to flash has an invalid header. Usually this happens when using a DNX module that supports an old DNX IFWI layout and trying to flash a DNX IFWI with a newer layout. If you see this error - match the versions of Intel® FIT used to generate the DNX IFWI and the DNX module.
Error code: 0x8000002e - Invalid partition	If you get this error when running a data clear command, it means there's no GPP4 and no RPMB partitions on the flash, so there's no data to be cleared.
Error code: 0x80000031-- RPMB not provisioned	If you get this error when trying to read data from RPMB it means that the RPMB isn't provisioned.
Error code: 0x80000035 - Image and descriptor mismatch	This happens when trying to flash a SPI IFWI that has different regions than the IFWI already flashed on the device. This flow isn't supported. SPI DNX only supports flashing IFWIs that have the exact same regions as the already flashed IFWI.
Error code: 0x80000039 - Invalid regions	This means your DNX IFWI doesn't match the NVM used. Usually this happens when hard straps are set to one NVM (e.g. SPI) and trying to flash an IFWI that is for a different NVM (e.g. UFS).
Error code: 0x8000003a - Unsupported storage device	Flash initialization failed. Check that your flash is connected properly to your platform, all the hard straps are set correctly and you



Error Code	Detail
	have all the needed reworks.
Error code: 0x80000043 - Invalid image layout	This means that the DNX IFWI you are trying to flash has an invalid layout. This happens when using a DNX module that supports a new DNX IFWI layout and trying to flash a DNX IFWI with an older layout. If you see this error - match the versions of Intel® FIT used to generate the DNX IFWI and the DNX module.
Error code: 0x80000045 - No DNX ifwi key in oem key manifest	This means that the DNX IFWI created doesn't support DNX since it misses the DNX IFWI usage in the OEM key manifest. The DNX IFWI usage needs to be added to the OEM key manifest in the IFWI.