UEIPAC G6 Intel Software Manual 5.2.0

October 2025 Edition

Contents

1 Introduction

The UEIPAC G6 (Intel) is UEI’s sixth generation of our UEIPAC product line.

Like all UEIPACs, the UEIPAC G6 Intel extends the capability of PowerDNA distributed data acquisition systems, allowing you to create programs that execute directly on Cube or RACK hardware. UEIPAC G6 Intel systems can be configured by UEI to run applications on any of the following operating systems:

- Windows 11 IoT Enterprise
- Rocky Linux 9.6
- RedHawk Linux RTOS

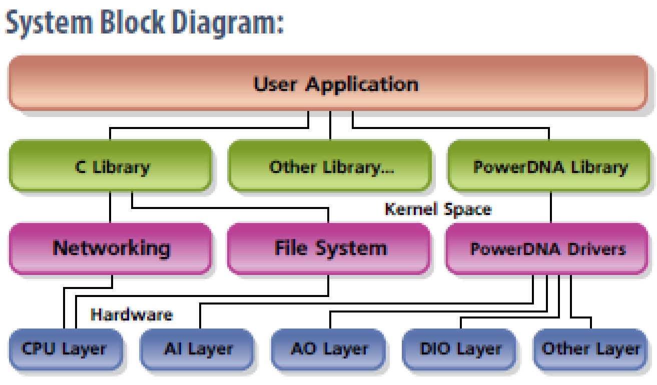
You can create standalone applications that do not require a host PC to control and monitor your hardware. Standalone applications can be developed directly on the UEIPAC or on a host PC. Alternatively, applications can be developed and run on a host PC.

The UEIPAC G6 is based on the Intel Atom x6425RE 64-bit CPU. By default, Windows and Linux installations reside in the onboard eMMC flash. This includes components of the operating system such as libraries, utilities, scripts, and daemons or Windows services.

After powering-up you have a ready to go UEIPAC Intel system running Windows or Linux. Access to the UEIPAC can be through the network or DisplayPort. DisplayPort allows for command shells via a GUI interface that can be directly accessed after connecting a USB keyboard and mouse and DisplayPort monitor. See “Configuring the UEIPAC” for information on how to connect to the UEIPAC Intel.

For Windows deployments, a Remote Desktop connection allows you to pass files by copy-paste between the UEIPAC Intel and a Windows host and provides seamless access to the Windows 11 Graphical User Interface via the host computer.

The UEIPAC can be configured to execute user applications after booting-up and can be configured for synchronization of I/O board clocks on multiple chassis using IEEE-1588 PTP or an external 1PPS reference. The UEIPAC Intel also provides Time Sensitive Networking (TSN) support. Intel® Time Coordinated Computing (TCC) is supported on Linux deployments.



For Windows deployments, user applications run as a regular Windows process. This gives you access to the Windows API provided by the system libraries, as well as any other library that can be compiled or installed (for example, libmodbus, openssl).

For Linux deployments, user applications run as a regular Linux process. This gives you access to the standard POSIX API provided by the GNU C runtime library (glibc), as well as any other library that can be compiled for Linux (for example, libxml, libaudiofile).

Supported development environments and languages for Windows systems include:

- Visual Studio 2010 to 2022
- C/C++
- Python
- LabVIEW

For more information developing applications on UEIPAC Intel systems, please see "Setting up a Development System" for more information.

1.1 Product Versions

The UEIPAC G6 Intel is currently available in a single option, the -40. A second generation, designated the -60 series, is being developed with an auxiliary PCIe interface. The UEIPAC G6 Intel -40 is an Intel x6425RE CPU with 8 GB RAM, 32 GB eMMC flash memory and a 4k DisplayPort video interface.

Table 1 provides a summary of feature differences between UEIPAC product versions.

Table 1 - Summary of UEIPAC Product Versions

Product	Summary of Features
UEIPAC PowerPC (UEIPAC -00 / -02 / -03)	• 10/100/1000Base-T Ethernet interface • 1 USB 2.0 port • Freescale MPC8347 PowerPC CPU • 1PPS/IEEE-1588 synchronization support[1] • Optional solid-state hard drives • Up to 256 MB RAM • 32 MB flash memory (-00 / -01/ -02) 128 MB flash memory (-03)
UEIPAC SoloX (G4) (UEIPAC -11 / -12)	• 10/100/1000Base-T Ethernet interface • 2 USB 2.0 ports • NXP i.MX6 SoloX Series ARM CPU (Cortex-A9 @ 1 GHz) • 1PPS/IEEE-1588 synchronization support[1] • Optional M.2 solid-state hard drives • 1 GB RAM • 8 GB flash memory (& U-boot QSPI flash) • HDMI support (UEIPAC SoloX -12 version only) • Optional GSM / Wireless support
UEIPAC Zynq (G5) (UEIPAC -33 / -34 / -3A)	• 3x 10/100/1000Base-T Ethernet interfaces • 1 USB 3.0 port • Xilinx Zynq UltraScale+ MPSoC (Quad core: Cortex-A53 & Arm Mali GPU) • 1PPS/IEEE-1588 synchronization support[1,2] • Optional M.2 solid-state hard drives • 4 GB RAM (-33 and -34 options) 2 GB RAM (-3A option) • 8 GB flash memory (U-boot, FPGA Logic, QSPI flash) • DisplayPort support

[1] 1PPS and IEEE-1588 synchronization support is described in the *PowerDNx 1PPS Sync Interface Manual*.

[2] IEEE-1588 PTP is currently only supported on NIC1/eth0 on Zynq

Product	Summary of Features
UEIPAC Intel (G6) (UEIPAC -40)	• Two independent Gigabit Ethernet ports • 1 USB 3.0 port, 1 USB 2.0 port • Quad-Core Intel x6425RE 64-bit CPU • 1PPS/IEEE-1588 synchronization support[3] • Optional M.2 solid-state hard drives • TSN protocol 802.1AS, 802.1Qbv, 802.1Qav, 802.1Qbu3 • TPM 2.0 security • 8 GB RAM • 32 GB on-board eMMC memory • DisplayPort support

[3] Only supported on NIC1

1.2 Usage of terms

Where applicable, we highlight differences between the earlier generations of the UEIPAC product vs the UEIPAC G6 for customers transitioning between the two.

- This manual provides descriptions of the UEIPAC G6, or **UEIPAC Intel**, which is based on the Intel Atom x6425RE 64-bit CPU.
- Previous generations of the UEIPAC products are described in Table 1 and based on the processor listed in the "Summary of Features" column. This manual will refer to those products as applicable:
    - **UEIPAC PowerPC**
    - **UEIPAC SoloX (UEIPAC G4)**
    - **UEIPAC Zynq (UEIPAC G5)**

2 Setting up a Development System

The UEIPAC G6 Intel development system is composed of the software tools necessary for users to develop applications targeting the Intel Atom x6425RE 64-bit CPU running one of the following operating systems:

- Windows 11 IoT Enterprise
- Rocky Linux 9.6
- RedHawk Linux RTOS (installs on top of Rocky Linux)

Users have the option of developing standalone applications that run directly on the Intel CPU or developing hosted applications that run on a Windows host or a Linux host. For standalone applications, development can be done on a separate PC or on the UEIPAC Intel CPU itself. Hosted applications are developed on the host PC. The development environment is the same whether development is done on the UEIPAC Intel or on a host PC.

Table 2 provides a summary of the supported development and runtime environments.

Table 2. UEIPAC Intel Supported Development and Runtime Environments

Development Environment	Runtime Environment
UEIPAC Windows	UEIPAC Windows
Windows PC	UEIPAC Windows
Windows PC	Hosted Windows (PDNA Server)
UEIPAC Linux	UEIPAC Linux
Linux PC	UEIPAC Linux
Linux PC	Hosted Linux (PDNA Server)

The UEIPAC Intel comes with PowerDNA Server preinstalled. PowerDNA Server implements the DAQBios protocol. This allows for development on a Windows or Linux host PC using development tools such as LabVIEW, Python, C++, C# and allows the host to run PowerDNA applications over Ethernet.

Additionally, UEI provides the PowerDNA Software Suite which can be downloaded from www.ueidaq.com/downloads. The PowerDNA Software Suite includes the following:

- Libraries for accessing PowerDNA data acquisition devices
- Header files
- Drivers
- PDNA Server
- Example programs
- PowerDNA Explorer (Windows systems)

2.1 Windows Development

Windows systems come with the UEIPAC Intel Runtime Installer preinstalled. This includes the PowerDNA Runtime Installer, which includes the libraries required for running applications. The UEIPAC Intel Runtime Installer also includes UEI drivers and PowerDNA Server.

To install the PowerDNA Software Suite for Windows, download the installer from www.ueidaq.com/downloads and run the installer (`PowerDNASoftware_5.x.x.x.exe`). The PowerDNA Software Suite includes the components listed in Table 3. On Windows systems, the components are installed in the folders listed in the table under the UEI installation folder. By default, the UEI installation folder is:

```
C:\Program Files (x86)\UEI
```

Table 3. PowerDNA Software Suite Components

PowerDNA Software Suite Component	Installation Folder (relative to UEI installation folder)
Libraries for accessing PowerDNA data acquisition devices	`.\PowerDNA\SDK\Lib`
Header files	`.\PowerDNA\SDK\includes`
Example programs for UEIPAC	`.\PowerDNA\SDK\Examples\UEIPAC`
Example programs for PowerDNA (hosted) mode	`.\PowerDNA\SDK\Examples\Visual C++`
PowerDNA Explorer	`.\PowerDNA\SDK\Applications`
UEI User Manuals	`.\PowerDNA\SDK\Documentation`

Install your preferred development tools on the UEIPAC Intel or host PC, e.g., Microsoft Visual Studio or mingw-w64.

Refer to Chapter 7, “Application Development”, for more information.

2.2 Linux Development

For Linux deployments, the UEIPAC G6 Intel default configuration is Rocky Linux 9.6. The UEIPAC G6 Intel can also be ordered with Concurrent Real-Time's RedHawk™ Linux RTOS. RedHawk Linux is ideally suited for applications where precise control and reliable data acquisition are essential. An extensive suite of features is provided to meet the specific requirements of real-time and mission-critical environments. To learn more about RedHawk Linux, visit the RedHawk Linux page on Concurrent Real-Time's website at https://concurrent-rt.com/products/software/redhawk-linux. RedHawk Linux is based on Rocky Linux.

Rocky Linux development tools include the following:

- GCC compiler targeting the UEIPAC Intel x6425RE 64-bit CPU
- GNU toolchain tools, such as `make`
- Standard Linux libraries, such as glibc

For Linux deployments, installing the PowerDNA Software Suite will provide source code and example programs. Files can be found in the locations listed in Table 4.

Table 4. PowerDNA Software Suite File Locations on Linux Systems

Source code for libraries and drivers	`/usr/local/src`
Examples	`/usr/local/examples`

Refer to Chapter 7, Application Development, for more information.

3 Configuring the UEIPAC

The UEIPAC Intel hardware runs as either a standalone or hosted system running Windows 11 IoT, Rocky Linux, or RedHawk Linux (installed on top of Rocky Linux).

Users can access the UEIPAC Intel using the network or via a DisplayPort connection with a DisplayPort monitor and USB keyboard and mouse. For network access you can use either the default IP address or configure IP addresses as required for your system.

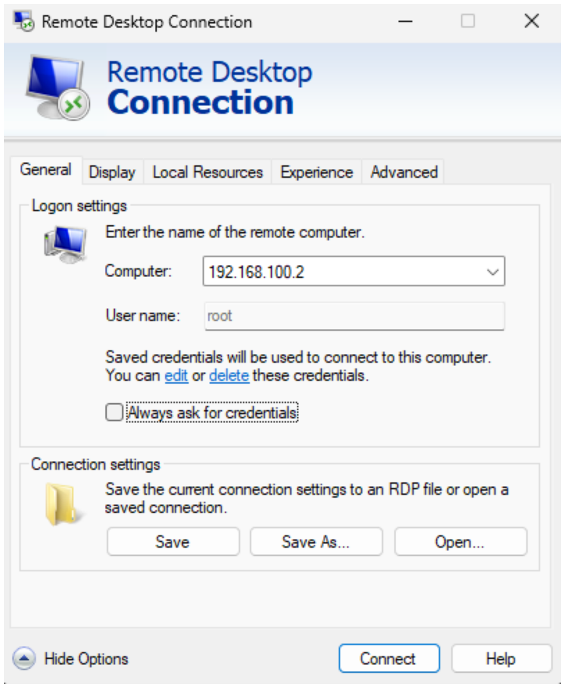
By default, Windows installations reside in the onboard eMMC flash. The default location for Linux installations is the onboard eMMC flash or the internal M.2 drive. This includes components of the operating system such as libraries, utilities, scripts, and daemons or Windows services.

This section provides information for configuring the UEIPAC network settings, boot device, and additional setup.

3.1 Connecting Using Remote Desktop (Windows)

A Remote Desktop connection allows you to pass files by copy-paste between the UEIPAC Intel and a Windows host and provides seamless access to the Windows 11 Graphical User Interface via the host computer. Use the following steps to connect to the UEIPAC Intel using a Remote Desktop connection:

1. Power up the UEIPAC.
   Connect the DC output of the 24 V power supply to the “Power In” connector on the UEIPAC Intel and connect the AC input on the power supply to the AC power source.
2. Connect the Ethernet port on the UEIPAC Intel to the Ethernet port on the host via the included cable.
3. From a Windows Host, find the Remote Desktop application using the Search bar.
4. Once Remote Desktop is open (Figure 1), enter the IP address of the UEIPAC Intel into the entry box for “Computer”. Remember, the factory default IP for NIC1 is 192.168.100.2 and for NIC2 is 192.168.101.2. Make sure that the host Ethernet address falls within the same subnet for the NIC that is connected.
5. Enter the “User name” (factory default user is “root”), then click “Open”.



Remote Desktop Connection

General | Display | Local Resources | Experience | Advanced

Logon settings

Enter the name of the remote computer.

Computer: 192.168.100.2

User name: root

Saved credentials will be used to connect to this computer. You can edit or delete these credentials.

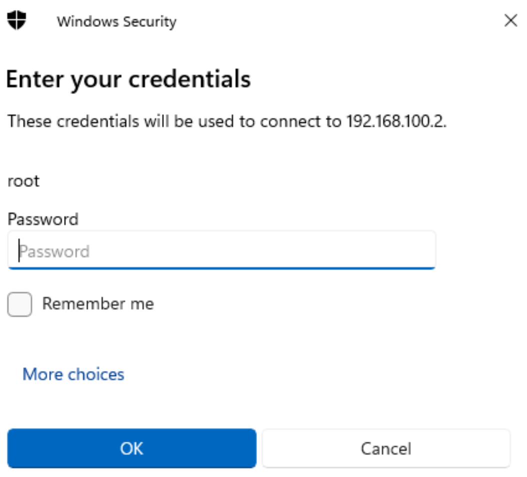
Always ask for credentials

Connection settings

Save the current connection settings to an RDP file or open a saved connection.

Figure 1. Remote Desktop Connection

6. When prompted, enter the password for default user “root” (Figure 2). The default password when shipped is “root”. Press “OK”.



Windows Security

Enter your credentials

These credentials will be used to connect to 192.168.100.2.

root

Password

Remember me

Figure 2. Enter Credentials

3.2 Connecting with SSH (Secure Shell)

The UEIPAC Intel comes with the OpenSSH SSH server preinstalled. The SSH protocol can be used to access a UEIPAC Intel system as follows:

1. Power up the UEIPAC.
   Connect the DC output of the 24 V power supply to the "Power In" connector on the UEIPAC Intel and connect the AC input on the power supply to the AC power source.
2. Connect the Ethernet port on the UEIPAC Intel to the Ethernet port on the host via the included cable.
3. Open a terminal window and type:
   ```
   ssh <username>@<IP address>
   ```
   Remember, the factory default user is "root", and the default IP address is 192.168.100.2 for NIC1 and 192.168.101.2 for NIC2.
4. When prompted for a password, enter the password for the user. The factory default user's password is "root".
   This will give access to a command prompt.
5. Type the command "`exit`" to logout.

You can avoid typing the password each time you log in using SSH keys:

1. Create private and public SSH keys on your host PC
   ```
   ssh-keygen -t rsa
   ```
2. Copy the public key to an `.ssh` directory on the UEIPAC as follows:

   for Windows (administrators group):
   ```
   scp /.ssh/id_rsa.pub \
     root@<IP-address>:C:/ProgramData/ssh/administrators_authorized_keys
   ```
   for Windows (non-administrators):
   ```
   scp /.ssh/id_rsa.pub \
     C:/Users/<Username>/.ssh/authorized_keys
   ```
   for Linux:
   ```
   scp ~/.ssh/id_rsa.pub \
     root@<IP-address>:/root/.ssh/authorized_keys
   ```
3. After this, you can log in to the UEIPAC Intel using SSH without a password.

NOTE: If your UEIPAC does not have the `.ssh` directory, `scp` will display an error message stating that the directory does not exist. If that happens, SSH onto the UEIPAC, create the `.ssh` directory and then type the `scp` command again on your host PC.

3.3 Connecting with DisplayPort

Use the following steps to connect directly to the UEIPAC Intel with a DisplayPort monitor and USB keyboard and mouse through the DisplayPort connector and one of the two USB ports.

1. Power up the UEIPAC.
   Connect the DC output of the power supply (24 VDC) to the Power In connector on the UEIPAC Intel and connect the AC input on the power supply to the AC power source.
2. Plug in the DisplayPort cable from the monitor into the DisplayPort connector on the CPU front panel.
3. Plug a USB keyboard/mouse pair into one of the USB ports near the DisplayPort connection.
4. If it is your first time logging in, log in as the factory-configured default user “root”, which can be accessed with default password “root”.
5. Once logged in as “root”, you can create system users and configure settings using either a terminal window or the GUI.

3.4 systemd Service Manager and runlevel Targets (for Linux Systems)

The following sections describe using systemd for configuring services and configuration of runlevel targets.

3.4.1 Configuring Services Using systemd

On Linux-based systems, the UEIPAC Intel uses the `systemd` daemon as its system and service manager. This is the first process that runs after the UEIPAC Intel boots up (PID 1).

The `systemctl` command is the main tool for managing `systemd`.

For more information about `systemd`, refer to the Rocky Linux site:

https://docs.rockylinux.org/books/admin_guide/16-about-sytemd

3.4.2 Configuring runlevel Targets (boot in GUI or non-GUI)

Rocky Linux on the UEIPAC Intel supports using different systemd target configurations to control what services, processes, and resources are used in the system.

These target configurations are represented as systemd target units, where predefined targets are mapped to a standard set of runlevels. Once the runlevel is configured, the system will boot in that operating state until the runlevel is changed again.

On the UEIPAC Intel, the default configuration is a non-graphical, multi-user interface (`multi-user.target`). To confirm the runlevel default for your system, enter the following:

```
# systemctl get-default
multi-user.target
```

To change what runlevel your system is configured with, use the `systemctl set-default` command:

```
~# systemctl set-default graphical.target
```

For more information about runlevel configuration, refer to the Rocky Linux site:

https://docs.rockylinux.org/books/admin_guide/16-about-sytemd/?h=runlevel#systemd-as-pid-1

3.5 Configuring the Network

This section provides an overview of how to configure the IP address, DHCP, and other network settings on the UEIPAC Intel. The UEIPAC Intel provides two independent Ethernet ports, NIC1 and NIC2. NIC 1 supports IEEE 1588, TSN, and TCC (TCC is supported on Linux deployments only). Ethernet port locations for the UEIPAC Intel are shown in Figure 3.

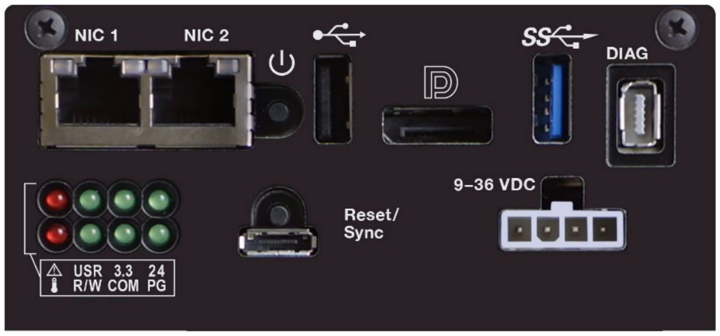


Figure 3. UEIPAC Intel Port Locations

UEIPAC Intel systems are configured at the factory with a static IP address to be part of a private network. The default IP address for NIC1 is 192.168.100.2. The default IP address for NIC2 is 192.168.101.2.

Note that both Ethernet ports should not be configured on the same subnet (for example, assigning NIC1 to 192.168.100.2 and NIC2 to 192.168.100.3). This will cause errors with the packet routing.

3.5.1 Configuring IP Addresses using UEI Scripts

UEI provides scripts to update the IP addresses of your UEIPAC Intel. The following sections provide information for updating IP addresses over Ethernet and using a DisplayPort connection, as well as additional information specific to Windows and Linux.

3.5.1.1 Running IP Setup Scripts over Ethernet

The scripts can be run over an Ethernet connection as follows:

1. Connect a PC directly to the NIC1 port of your UEIPAC Intel using an Ethernet cable.
2. Record the current IP address of the PC network adapter.
3. Configure the PC network adapter IP address to 192.168.100.x. Note that x can be 1 to 254 but x cannot be 2 since that is the default.

4. At a command prompt, enter "`ssh 192.168.100.2`".
5. For Linux, enter "`setipeth0 <new IP address>`" to change the IP address for NIC1.
   For Windows, use "`setip1 <new IP address>`" to change the IP address for NIC1.
6. Configure the PC network adapter IP address to its former IP address.
7. At a command prompt, use the `ssh` command with the newly assigned IP address for the UEIPAC Intel.

   NOTE: The instructions above can also be used to change the IP address of NIC2. NIC2 comes with a default IP address of 192.168.101.2.
   For Linux, use `setipeth1` to change the IP address for NIC2.
   For Windows, use `setip2` to change the IP address for NIC2.

3.5.1.2 Running IP Setup Scripts using DisplayPort

To run the scripts for updating the IP Addresses of your UEIPAC Intel using a DisplayPort (DP) connection, perform the following steps.

1. Connect a DP monitor to the DP port and a USB keyboard and mouse through a USB hub to the USB port.
2. For Linux users, enter "`setipeth0 <new IP address>`" at a command prompt to change the IP address for NIC1. Use `setipeth1` to change the IP address for NIC2.
   For Windows users, enter "`setip1 <new IP address>`" at a command prompt to change the IP address for NIC1. Use `setip2` to change the IP address for NIC2.

3.5.1.3 Additional Information for Windows Users

Windows users can also use command prompt utilities such as `netsh` and `ipconfig` or use the Windows GUI by navigating to Settings > Network & internet > Ethernet and selecting the Edit button next to "IP assignment" to edit the IPv4 address.

3.5.1.4 Additional Information for Linux Users

In addition to the scripts provided by UEI, Linux users can use the commands listed in this section for updating the IP address of the NIC ports. Use the `lshw` command to obtain the logical names of NIC1 and NIC2. For example:

```
lshw -c network
```

You can make a temporary change to the IP address of NIC1 or NIC2 by using the `ip` command or make a persistent change by using the `nmcli` command.

For example, to set an IP address with the `ip` command, enter the following, where `enp0s20f0u1` is the logical name of the NIC port (NIC2 in this example). This change will not persist through a reboot.

```
[root@localhost ~]# ip addr delete 192.168.101.2/24 dev enp0s20f0u1

[root@localhost ~]# ip addr add 192.168.101.4/24 brd + dev enp0s20f0u1
```

To view the IP address of the NIC ports, use the `ip` command as follows (omit the `-br` option for more detailed output):

```
[root@localhost ~]# ip -br addr show
```

To make changes persistent on a reboot, do the following to configure NIC1 (`nmtui` can also be used):

```
~# nmcli connection modify eno1 ipv4.address "192.168.100.4/24"
```

UEI also provides convenience scripts for setting a persistent IP address within a 255.255.255.0 network.

To make changes persistent for NIC1:

```
~# setipeth0 <IP address>
```

or:

```
~# setipeth <IP address>
```

To make changes persistent for NIC2:

```
~# setipeth1 <IP address>
```

More information on network configuration can be found in the Rocky Linux documentation:

https://docs.rockylinux.org/guides/network/basic_network_configuration/

and on the NetworkManager website:

https://networkmanager.dev/docs/

3.5.2 Changing the Default Packet Size (MTU)

The default MTU size is 1500 bytes. You can check the current MTU setting using:

```
~# ip link show
```

You can change the MTU parameter for an Ethernet port with the `ip link set` command. For example, to change MTU for NIC1 to 1450 bytes:

```
~# ip set eno1 mtu 1450
```

3.5.3 Configuring Dynamic IP Address (Using a DHCP Server)

On Linux systems, if you are connected to a DHCP server, you can configure the UEIPAC Intel to automatically fetch an IP address when it boots up.

For example, to change eno1, do the following:

1. Switch from manual to auto:
   ```
   nmcli con mod eno1 ipv4.method auto
   ```
2. Remove the static IP address:
   ```
   nmcli con mod eno1 ipv4.address "ipv4.gateway"
   ```
3. Restart the interface:
   ```
   nmcli con down eno1
   nmcli con up eno1
   ```

After restarting the interface, you can learn the dynamically assigned IP address by connecting to the UEIPAC Intel over the DisplayPort connection and use a USB keyboard / mouse, and type `ip` at the Linux prompt.

3.6 Configuring Time and Date

The time, date, and time zone can be set as described in the following sections. UEIPAC Intel systems can also be configured to use Network Time Protocol (NTP).

3.6.1 Windows Systems

For Windows systems, the time, date, and time zone can be set in Windows Settings by users with an Administrator account type.

1. Open Windows Settings
2. In the navigation pane on the left-hand side of the display, select “Time & language”.
3. Select “Date & time”.

From the “Date & time” screen, the time zone can be selected. Turn off “Set time automatically” and select the Change button to manually set the date and time.

For information on using Windows Time Service and NTP to synchronize the UEIPAC Intel clock with a time server, refer to

https://learn.microsoft.com/en-us/windows-server/networking/windows-time-service/windows-time-service-tools-and-settings?tabs=config

3.6.2 Linux Systems

This section provides an overview of how to configure time and date settings on Linux systems. For more information about `systemd` time and date control, refer to the Rocky Linux man pages:

https://docs.rockylinux.org/guides/network/librenms_monitoring_server/?h=timedatectl#set-timezone

The time, date, and NTP services, are configured with the `timedatectl` command. You can type `timedatectl --help` for usage.

3.6.2.1 Changing the Time and Date

The UEIPAC Intel is equipped with a real-time clock (RTC) chip that preserves the date and time settings when the UEIPAC Intel is not powered. By default, the date is set to the current date and time in the UTC (GMT) time zone.

To print the current date and time, you can enter `date` at the Linux prompt. You can also set the date using `date` and write it to RTC (make persistent) with `hwclock`:

```
# date --set="05/28/2019 13:48:00"
# hwclock -w
```

Alternatively, to see the full systemd time and date configuration, enter `timedatectl`:

```
# timedatectl
      Local time: Fri 2019-04-05 15:50:03 UTC
  Universal time: Fri 2019-04-05 15:50:03 UTC
        RTC time: Fri 2019-04-05 15:50:05
       Time zone: Universal (UTC, +0000)
 Network time on: no
NTP synchronized: no
 RTC in local TZ: no
```

You can update the current date and time on your system and store it in your RTC using the `timedatectl` command with the `set-time` argument.

As an example, `timedatectl set-time "2019-06-05 01:02"` sets the new date to June 5, 1:02 AM.

Note: If NTP synchronization is enabled, you cannot set the time manually:

```
# timedatectl set-time "2019-04-05 01:02"
Failed to set time: Automatic time synchronization is enabled
```

3.6.2.2 Connecting to an NTP Server

By default, NTP synchronization is disabled on UEIPAC Intel.

You can turn NTP synchronization on with the `timedatectl` command:

```
# timedatectl set-ntp ON
# timedatectl status
      Local time: Fri 2019-04-05 19:05:36 UTC
  Universal time: Fri 2019-04-05 19:05:36 UTC
        RTC time: Fri 2019-04-05 19:05:37
       Time zone: Universal (UTC, +0000)
 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
```

NIC 1 should be connected to a network that has access to an NTP server. When starting, `systemd-timesyncd` reads the configuration file located at `/etc/chrony.conf`. You can add or change time servers by editing the `chrony.conf` configuration file.

3.6.2.3 Changing the Time Zone

To print a list of possible time zones, enter the following:

```
~# timedatectl list-timezones
```

To set the time zone, copy the name from the listed zones, and enter that as the set parameter. For example, to set the time zone to Eastern time in the USA:

```
~# timedatectl set-timezone "America/New_York"
~# timedatectl

      Local time: Fri 2019-04-05 15:21:37 EDT
  Universal time: Fri 2019-04-05 19:21:37 UTC
        RTC time: Fri 2019-04-05 19:21:38
       Time zone: America/New_York (EDT, -0400)

 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
```

To reset it to UTC, pass `UTC` instead of `"America/New_York"`

3.7 Configuring Boot Media

The UEIPAC Intel BIOS configuration screen can be used to select the preferred boot device. Select your boot device as follows:

1. Power up the UEIPAC Intel.
2. While the system is booting, repeatedly press the Delete key.
3. When the Aptio Setup screen is displayed, select the Boot tab.
4. Select the preferred boot media.

   For Windows systems, the boot device options are:

   - Onboard eMMC Storage
   - NVMe Storage (M.2 drive)

   By default, UEIPAC Intel Windows systems are configured to boot from eMMC.

   For Linux systems, the boot device options are:

   - USB
   - Onboard eMMC Storage
   - NVMe Storage (M.2 drive)

   By default, UEIPAC Intel Linux systems are configured to boot from USB. If no USB drive is found, the system will boot from eMMC.

3.8 Adding User Accounts and Changing Passwords

User accounts and passwords for Windows and Linux systems can be managed as described in the following sections.

3.8.1 Windows Systems

For Windows systems, users can be added as follows. You must be logged on with an Administrator account type to create user accounts.

1. Open Windows Settings
2. In the navigation pane on the left-hand side of the display, select "Accounts".
3. Under Account, select "Other Users".
4. In "Accounts > Other Users", select the "Add account" button.
5. Follow the instructions to add a user account.

Windows users can change their own password as follows.

1. Open Windows Settings
2. In the navigation pane on the left-hand side of the display, select "Accounts".
3. Under Account, select "Sign-in options".
4. In "Accounts > Sign-in options", select "Password".
5. Select the "Change" button.
6. Enter your current password.
7. Enter and confirm your new password.

3.8.2 Linux Systems

For Linux systems, follow the instructions in this section to add users or change passwords.

When logged in as root, use standard Linux commands to set up user accounts:

1. Add the new user:

   ```
   ~# adduser <username>
   ```

2. Configure user groups:

   ```
   ~# usermod -a -G root,sudo,adm,lp,dialout,audio,video,systemd-
   journal,systemd-network,netdev <username>
   ```

When logged in as any user and you want to update your password, use the `passwd` command.

For example, you can change the root password when you are logged in as root by entering `passwd` at the Linux prompt and then entering your new password two times when prompted. After that, you can log out and log in with your new password.

NOTE: UEI provides a non-root user account in the Rocky Linux distribution: (username `uei`, and password `uei`).

The non-root user accounts have restricted access and require sudo privilege for certain activities. Many commands in /sbin are restricted. For example:

- Executing commands in `/sbin`: `sudo reboot`

Additionally, to update your path to include `ifconfig`, `reboot`, and other commands found in `/sbin`, enter the following:

```
~# export PATH=”$PATH:/sbin”
```

4 Transferring Files

Both Windows and Linux users can use the SSH protocol to transfer files between your host PC and the UEIPAC Intel. The OpenSSH server is pre-installed and enabled by default.

For Windows users, the UEIPAC Intel can be set as a shared folder. A connection to the shared folder can then be made from the Windows host PC. This allows folders and files to be easily transferred using drag and drop. Windows users can also use a third-party application such as MobaXterm which can work as SSH server/client.

For Linux or Windows users using SSH, the `scp` or `sftp` command can be used to transfer files between your PC and the UEIPAC.

To send a file via scp to the UEIPAC Intel, enter:

```
scp <source file path on PC> root@192.168.100.2:<destination path on
UEIPAC>
```

To retrieve a file from the UEIPAC, enter:

```
scp root@192.168.100.2:<source file path on UEIPAC> <destination path
on PC>
```

Use `sftp <UEIPAC IP address>` to transfer one or multiple file(s) using `sftp`.

4.1 Storage Devices for Linux Systems

The following storage device names are applicable to Linux systems:

- eMMC flash is device name `/dev/mmcblk0`
- USB devices are `/dev/sdxn`, where x is a for the first drive and b for the second drive. n is the partition number.
- M.2 SSD devices are `/dev/nvme*`

5 Connecting USB Storage Devices

The UEIPAC Intel provides one USB 3.0 port and one USB 2.0 port for external storage. Both ports use a USB type A connector. The following sections provide information for mounting and formatting USB devices on Linux systems. USB devices on Windows systems simply appear as a new drive letter in File Explorer.

5.1 Mounting USB Mass Storage Device in Linux Systems

UEIPAC Intel Linux systems support USB mass storage devices that comply with the USB mass storage device class and are formatted with one of the following formats: FAT, EXT2, EXT3, EXT4.

USB devices can be mounted with the following command:

```
~# mount /dev/sdxn/mnt
```

Alternatively, the following command can also be used;

```
~# udisksctl mount -b /dev/sdxn
```

This will mount the USB device under:

```
/run/media/<username>/<label>
```

Note the device node name assigned to this USB device is in the format “sdxn”:

- x is **a** for the first drive, **b** for the second.
- n is the partition number.

You can check the mount point using the `df` command:

```
~# df -h

Filesystem      Size  Used Avail Use% Mounted on
/dev/root       6.0G  1.1G  4.7G  19% /
devtmpfs        342M  4.0K  342M   1% /dev
tmpfs           502M     0  502M   0% /dev/shm
tmpfs           502M  8.8M  493M   2% /run
tmpfs           502M     0  502M   0% /sys/fs/cgroup
tmpfs           502M   36K  502M   1% /tmp
tmpfs           502M   44K  502M   1% /var/volatile
/dev/mmcblk2p1  8.0M  5.7M  2.4M  71% /run/media/mmcblk2p1
tmpfs           101M     0  101M   0% /run/user/0
/dev/sda1       7.5G  4.0K  7.5G   1% /run/media/sda1
```

USB devices are mounted in the `/run/media` directory by default, and the files are accessible under that `/run/media` directory.

To unmount, use the following command:

```
~# umount /run/media/sda1
```

If you need to format the device, refer to Section 5.2.

5.2 Formatting Storage Devices in Linux Systems

The following section provides instructions for formatting a storage device using the `mke2fs` utility on a Linux system.

The following procedure is an example of formatting a removable USB device (`/dev/sdb`) as a single partition with an EXT3 filesystem on a Linux host (the same steps can be used for other devices; however, the device name and mount point may be different):

1. Unmount the USB (if mounted):
   ```
    $ sudo umount /dev/sdb1
   ```
   NOTE: You may need to unmount multiple partitions (e.g., `umount /dev/sdb1`, `umount /dev/sdb2`, etc.)

2. Run **fdisk**

   Erase all partitions from the device (`d`) and create one primary partition (`n`) using all the available space on the card, and write it to hardware (`w`) :
   ```
   $ sudo fdisk /dev/sdb

   Welcome to fdisk (util-linux 2.27.1).
   Changes will remain in memory only, until you decide to
   write them.
   Be careful before using the write command.

   Command (m for help): d
   Selected partition 1
   Partition 1 has been deleted.

   Command (m for help): n
   Partition type
      p   primary (0 primary, 0 extended, 4 free)
      e   extended (container for logical partitions)

   Select (default p): p
   Using default response p.
   Partition number (1-4, default 1):
   First sector (2048-30605311, default 2048):
   Last sector, +sectors or +size{K,M,G,T,P} (2048-30605311, default
   30605311):
   ```

```
Created a new partition 1 of type 'Linux' and of size 14.6 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

3. Unmount the device again (if necessary)
   ```
   $ sudo umount /dev/sdb1
   ```
4. Format the USB device
   The device node associated with the partition we just created is "`/dev/sdb1`".
   Format this new partition with **mke2fs** (-j option sets file system type to ext3):

```
$ sudo mke2fs -j /dev/sdb1

mke2fs 1.42.13 (17-May-2025)
Creating filesystem with 3825408 4k blocks and 956592 inodes
Filesystem UUID: e42f53a6-5b6e-4b5e-94c4-b88dad9b999a
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736,
1605632, 2654208

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

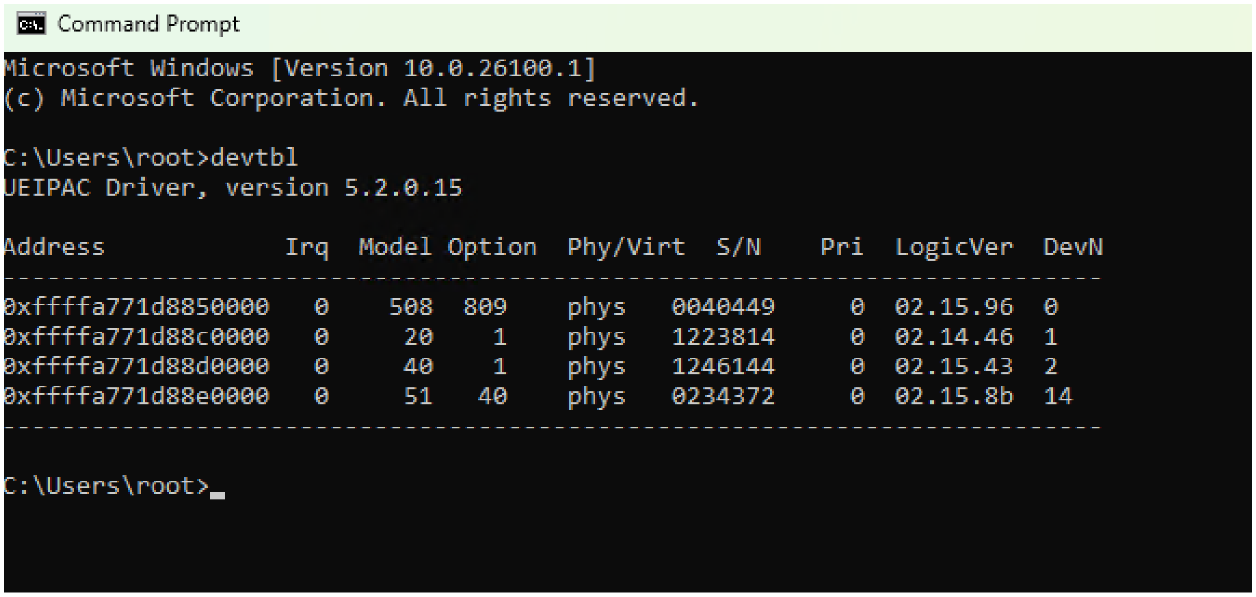
5. Mount the device (if necessary)
   ```
   $ mount /dev/sdb1 /mnt
   ```

You can follow the same instructions to format other storage devices using a Linux PC.

6 Testing the I/O Cards

6.1 devtbl

The `devtbl` command can be run from the command line in Windows or Linux. Running the command prints a list of the I/O cards that are detected in the module. Figure 4 shows an example of `devtbl` output on a Windows system.



```
Microsoft Windows [Version 10.0.26100.1]
(c) Microsoft Corporation. All rights reserved.

C:\Users\root>devtbl
UEIPAC Driver, version 5.2.0.15

Address             Irq  Model Option  Phy/Virt  S/N     Pri  LogicVer  DevN
---------------------------------------------------------------------------
0xffffa771d8850000   0    508   809    phys   0040449    0  02.15.96  0
0xffffa771d88c0000   0     20     1    phys   1223814    0  02.14.46  1
0xffffa771d88d0000   0     40     1    phys   1246144    0  02.15.43  2
0xffffa771d88e0000   0     51    40    phys   0234372    0  02.15.8b  14
---------------------------------------------------------------------------

C:\Users\root>_
```

Figure 4. **devtbl** Output

6.2 Run Examples

UEI provides example programs for both Windows and Linux UEIPAC Intel systems. Example programs can be compiled as is or they can be edited and compiled and then executed on either the host PC or directly on the UEIPAC Intel. provided with a makefile on your host PC or UEIPAC.

For Windows systems, use `CMake` to generate a makefile or Visual Studio project. The default location for the example programs is:

```
C:\Program Files (x86)\UEI\PowerDNA\SDK\Examples\UEIPAC
```

for running examples on the UEIPAC Intel or

```
C:\Program Files (x86)\UEI\PowerDNA\SDK\Examples\Visual C++
```

for PowerDNA mode.

For Linux systems, use `CMake` to generate a makefile. The example programs are installed in:

```
/usr/local/examples
```

You can create build files and build applications using the following steps:

1. Navigate to an example directory, for example:
   ```
   cd Sample207
   ```
2. Create build directory:
   ```
   mkdir build
   ```
3. Enter the build directory:
   ```
   cd build
   ```
4. Generate the build files:
   ```
   cmake ..
   ```
5. Build the application:
   ```
   cmake --build .
   ```
6. On Windows systems, this will create a debug version of the application in the Debug directory. For example, in Windows: `Debug\Sample207.exe`. On Linux systems, the application will be created in the `build` directory.

There is at least one example for each supported I/O card. Code examples are named “SampleXXX” (where XXX is the model ID of each I/O card).

When running an example program, the example, by default targets the first I/O card (device 0). You can change the device number, frequency, and enabled channels using command line options.

The following are a few of the available options:

```
-h : displays help
-d n: selects the device to use (default: 0)
-f n.nn: sets the rate of the DAQ operation (default: 1000 Hz)
-c "x,y,z,..." : selects the channels to use (default: channel 0)
```

For example, the following command runs the AI-207 test program using device 2 and channels 1, 2,and 7:

```
~# ./Sample207 -d 2 -c "1,2,7"
1 device(s) specified: 2
3 channel(s) specified: 1 2 7
 0: ch1 bdata 310c7efc fdata -7.519827V
 0: ch2 bdata 310f8140 fdata 7.524481V
 0: ch7 bdata 310c7efc fdata -7.519827V

 1: ch1 bdata 310c7efb fdata -7.519903V
 1: ch2 bdata 310f813c fdata 7.524176V
 1: ch7 bdata 310c7efe fdata -7.519674V
...
```

All examples are configured to stop when they receive the SIGINT signal. You can send this signal by typing CTRL+C. Alternatively, if the program runs in the background, Windows users can use the `taskkill` command from the Windows Command Prompt and Linux users can use `killall -SIGINT`.

6.3 PowerDNA Server

PowerDNA Server emulates the behavior of a PowerDNA IO module running the standard DAQBIOS firmware. It emulates a subset of the DAQBIOS protocol so that the UEIPAC can be accessed from PowerDNA Explorer or the PowerDNA C API. It works in immediate, RTDMAP and VMAP modes. ACB and Messaging modes are not supported.

6.3.1 Windows Systems

On Windows systems, PowerDNA Server runs as a service that can be enabled or disabled via the Windows Services application which can be accessed through the Windows Control Panel:

```
Control Panel > System and Security > Windows Tools > Services
```

In the list of services, select and right-click "`UEI Pdnaserver`" for a menu of available options, e.g., Start, Stop, etc.

6.3.2 Linux Systems

On Linux deployments, PowerDNA Server is automatically started by `inetd`. The `inetd` daemon monitors incoming packets and automatically starts the appropriate server.

If you need to stop the PowerDNA Server enter

```
~# systemctl stop inetd.service
```

To start again enter

```
~# systemctl start inetd.service
```

To prevent the PowerDNA Server from automatically starting, enter:

```
~# systemctl stop inetd.service
~# systemctl disable inetd.service
```

To reenable PowerDNA Server, enter:

```
~# systemctl enable inetd.service
~# systemctl start inetd.service
```

To prevent the PowerDNA Server from automatically starting at boot, edit `/etc/inetd.conf` and comment out the line

```
pdnaserver dgram udp wait root /usr/bin/pdnaserver pdnaserver -s 0 i lo
```

Then restart `inetd` by entering

```
~# /etc/init.d/busybox-inetd restart
```

6.4 PowerDNA Explorer

PowerDNA Explorer is a GUI-based application for interacting with your UEI Intel system. Also included are the libraries needed to run any applications developed for your UEIPAC Intel system.

PowerDNA Explorer capabilities include displaying diagnostic data for the CPU / Power module (temperatures, voltage supply measurements, error conditions), reporting serial numbers and versioning information, and providing the ability to configure I/O boards and read and write data.

The PowerDNA Software Suite provides a full version of PowerDNA Explorer along with UEI header files and example programs. The PowerDNA Software Suite can be downloaded from www.ueidaq.com/downloads and installed by running `PowerDNASoftware_5.x.x.x.exe`.

For more information about the features and usage of PowerDNA Explorer, refer to the *UEIPAC Intel Hardware Manual*.

7 Application Development

Application development for the UEIPAC Intel can be done on a Windows or Linux host computer or directly on the UEIPAC Intel.

7.1 PowerDNA Software Suite

The PowerDNA Software Suite implements the API used to program the PowerDNA I/O cards and includes PowerDNA Explorer, header files, and example programs. See Chapter 2, “Setting up a Development System” for information on PowerDNA Software Suite installation and directory structure.

For Windows systems, install the PowerDNA Software Suite for Windows. Supported development environments and languages include:

- Visual Studio 2010 to 2022
- C/C++
- Python
- LabVIEW

For Linux systems, install the PowerDNA Software Suite for Linux on your development environment, and use the development tools that are provided with Rocky Linux such as Python, gcc, make, etc.

The PowerDNA Software Suite additionally allows you to control remote UEIPAC Intel IO modules from the UEIPAC the same way you would from a host PC running Windows or Linux.

The PowerDNA API uses the IP address specified in the function `DqOpenIOM()` to determine whether you wish to access the I/O cards local to the UEIPAC or “remote” I/O cards installed in a remote I/O module.

Set the IP address to the loopback address “127.0.0.1” and the API will access the “local” I/O cards on the UEIPAC Intel itself.

The PowerDNA API implements the data acquisition modes listed in Table 5 to communicate with the I/O cards.

Table 5. PowerDNA Data Acquisition Modes

Mode	Description
Immediate (Point-by-point))	This is the easiest mode for point-by-point input/output on all I/O cards. It also is the least efficient because it requires one call for each incoming and/or outgoing request. You cannot achieve maximum performance with this mode Immediate mode examples are named "SampleXXX"
Real-Time Data Mapping (RtDMAP)	This is the most efficient mode for point-by-point input/output on AI, AO, DIO and CT cards. Incoming and outgoing data from/to multiple I/O cards are all packed in a single call. Note that RtDMAP mode replaces the legacy DMAP mode, which is not supported on UEIPAC Intel. RtDMAP mode examples are named "SampleRtDMapXXX" for PowerDNA mode. For UEIPAC mode, RtDMAP mode examples are named "SampleDMapXXX".
Asynchronous Data Mapping (ADMAP)	Allows access to input I/O cards at speeds up to 10 kHz, transferring data in one call when a when a user-programmable period of time has elapsed. ADMAP mode examples are named "SampleADMapXXX"
Buffered (ACB)	This mode is only available over Ethernet and **not** supported for communication with local I/O cards installed on the UEIPAC Intel. (For reference only, it allows access to AI, AO, DIO and CT I/O cards at full speed. It is designed to correct communication errors that might happen on the network link. The error correction mechanism will cause issues with real-time deadlines.) ACB mode examples are not included with the UEIPAC Intel PowerDNA Software Suite (as a reference for hosted applications, they are named "SampleACBXXX").

Mode	Description
Messaging	This mode is only available over Ethernet and **not** supported for communication with local I/O cards installed on the UEIPAC Intel. (For reference only, this mode allows access to messaging layers (serial, CAN, ARINC-429) at full speed. It is designed to correct communication errors that might happen on the network link. The error correction mechanism will cause issues with real-time deadlines) (As a reference for hosted applications, messaging mode examples are named “SampleMsgXXX”)
Variable Size Data Mapping (VMAP)	Allows access to all I/O cards at full speed, transferring incoming and outgoing data in buffers in one call. VMAP mode examples are named “SampleVMapXXX”
Asynchronous Variable Size Data Mapping (AVMAP)	Allows access to input I/O cards at full speed, transferring data in buffers in one call upon a watermark level or trigger event. AVMAP mode examples are named “SampleAVMapXXX”
Asynchronous	Allows I/O cards to asynchronously notify the user application upon hardware events Asynchronous mode examples are named “SampleAsyncXXX”

The UEIPAC Intel only supports the immediate (also known as “point-by-point”), DMAP, VMAP, AVMAP, and ADMAP modes to control the “local” I/O cards. Async mode is supported for selected I/O cards.

The other modes (ACB and MSG) are designed to work over Ethernet and have built-in error correction which is not needed on the UEIPAC Intel. You can, however, use those modes to control “remote” I/O cards installed in I/O modules that run the DAQBios firmware over the network. The remote I/O cards cannot be installed in a UEIPAC system. Table 6 lists the I/O modes supported by the UEIPAC Intel.

Table 6. UEIPAC I/O Modes

I/O Mode	Supported on UEIPAC (Local I/O Cards)
Immediate	Yes
ACB	No
DMAP	Yes
ADMAP	Yes
MSG	No
VMAP	Yes
AVMAP	Yes
Asynchronous Event	*Select I/O cards (refer to example code)

7.1.1 PowerDNA APIs

The following section details the subset of PowerDNA APIs available when running your program on a UEIPAC Intel system.

Refer to the *PowerDNA API Reference Manual* to get detailed information about each API.

7.1.1.1 Initialization, Miscellaneous API

The following APIs are used to initialize the library, obtain a handle on the kernel driver and perform miscellaneous tasks such as translating error code to readable messages.

- DqInitDAQLib
- DqCleanUpDAQLib
- DqOpenIOM
- DqCloseIOM
- DqTranslateError
- All DqCmd*** APIs

7.1.1.2 Immediate Mode API

Immediate Mode APIs are used to read/write I/O cards in a software-timed fashion. They are designed to provide an easy way to access I/O cards at a non-deterministic pace.

Each I/O card comes with its own set of immediate mode APIs. For example, you will use the DqAdv201*** APIs to control an AI-201.

Most DqAdvXYZ*** APIs, where XYZ is the model number of a supported I/O card, are supported on the UEIPAC.

7.1.1.3 RtDMAP API

In RtDMAP mode, the UEIPAC continuously refreshes a set of channels that can span multiple I/O cards at a specified rate paced by a hardware clock.

Values read from or written to each configured channel are stored in an area of memory called the DMAP. At each clock tick, the firmware synchronizes the DMAP values with their associated physical channels.

Supported APIs that use RTDMAP mode are called DqRtDmap***.

The following is a quick tutorial on using the RtDMAP API (handling of error codes is omitted):

1. Initialize the DMAP to refresh at 1000 Hz:

   ```
   DqRtDmapInit(handle, &dmapid,1000.0);
   ```

2. Add channel 0 from the first input subsystem of device 1:

   ```
   chentry = 0;
   DqRtDmapAddChannel(handle, dmapid, 1, DQ_SS0IN, &chentry, 1);
   ```

3. Add channel 1 from the first output subsystem of device 3:

   ```
   chentry = 1;
   DqRtDmapAddChannel(handle, dmapid, 3, DQ_SS0OUT, &chentry, 1);
   ```

4. Start all devices that have channels configured in the DMAP:

   ```
   DqRtDmapStart(handle, dmapid);
   ```

5. Update the value(s) to output to device 3:

   ```
   outdata[0] = 5.0;
   DqRtDmapWriteScaledData(handle, dmapid, 3, outdata, 1);
   ```

6. Synchronize the DMAP with all devices:

```
DqRtDmapRefresh(handle, dmapid);
```

7. Retrieve the data acquired by device 1:

```
DqRtDmapReadScaledData(handle, dmapid, 1, indata, 1);
```

8. Stop the devices and free all resources:

```
DqRtDmapStop(handle, dmapid);

DqRtDmapClose(handle, dmapid);
```

7.1.1.4 ADMAP API

In ADMAP mode, the UEIPAC continuously refreshes a set of channels that can span multiple I/O cards.

Each input I/O card is programmed to acquire data from its internal FIFO at a rate paced by its hardware clock.

The user application is notified when a user-programmable period of time has elapsed and data is ready for retrieval.

The content of all the I/O cards' FIFOs is accessed in one operation.

Supported APIs for using ADMAP mode are DqRtDmap*** and DqRtAXmap***.

The following is a quick tutorial on using the ADMAP API (handling of error codes is omitted):

1. Initialize the ADMAP to refresh at 1000 Hz:

   ```
   DqRtDmapInit(handle, &dmapid,1000.0);
   ```

2. Add channel 0 from the first input subsystem of device 1:

   ```
   chentry = 0;
   DqRtDmapAddChannel(handle, dmapid, 1, DQ_SS0IN, &chentry, 1);
   ```

3. Add channel 1 from the first output subsystem of device 3:

   ```
   chentry = 1;
   DqRtDmapAddChannel(handle, dmapid, 3, DQ_SS0OUT, &chentry, 1);
   ```

4. Start all devices that have channels configured in the DMAP:

   ```
   DqRtAXMapStart(handle, dmapid, DQ_AVMAP_TYP_TIME, params.frequency,
         DQ_AVMAP_CLK_CFG(SYNC_LINE, CLOCK_DIV), 0);
   ```

5. Allocate slots in the DMAP:

   ```
   DqRtAXMapSlotAllocate(handle, TRUE, dmapid, 0);
   ```

6. See if there is an asynchronous packet.

   ```
   DqRtDmapRefreshInputsExt(handle, dmapid, &counter, &timestamp);
   ```

7. Start asynchronous operation.

   ```
   DqRtAXMapEnable(handle, TRUE);
   ```

8. Continue to check for asynchronous packet and read data until done with the following two functions.:.

```
DqRtDmapRefreshInputsExt(handle, dmapid, &counter, &timestamp);

DqRtDmapReadScaledData(handle, dmapid, params.device, fdata,
      params.numChannels);
```

9. Stop the devices and free all resources:

```
DqRtAXMapEnable(handle, FALSE);

DqRtDmapStop(handle, dmapid);

DqRtDmapClose(handle, dmapid);
```

7.1.1.5 VMAP API

In VMAP mode, the UEIPAC Intel continuously acquires/updates data in buffers.

Each I/O card is programmed to acquire/update data to/from its internal FIFO at a rate paced by its hardware clock.

The content of all the I/O cards' FIFOs is accessed in one operation.

Supported APIs to use VMAP mode are DqRtVmap***.

The following is a quick tutorial on using the RTVMAP API (handling of error codes is omitted):

1. Initialize the VMAP to acquire/generate data at 1 kHz:

   ```
   DqRtVmapInit(handle, vmapid, 1000.0);
   ```

2. Add channels from the first set of input ports of a messaging layer (serial, 1553, ARINC, etc.) at device 0 as follows:

   ```
   //configure input ports 0, 1, 2, and 3
   int channels[] = {0, 1, 2, 3 };
   int flags[] = {0, 0, 0, 0 };

   DqRtVmapAddChannel(handle, vmapid,0,DQ_SS0IN,channels,flags,4);
   ```

3. Add channels from the first analog input, analog output, and/or digital I/O card (analog input (input subsystem) in this example), for device 1, as follows:

   ```
   //initialize a VMAP channel, which for AI/AO/DIO cards is a
   // virtual channel streaming all physical channels (interleaved)

   int vmapChannel = 0; // the actual value doesn't matter
   int flag = 0;

   DqRtVmapAddChannel(handle, vmapid, 1, DQ_SS0IN, &vmapChannel, &flag,
   1);

   //add the input channels 0, 1, 2, 3, and 4
   int channels[] = {0, 1, 2, 3, 4 };
   int flags[] = {0, 0, 0, 0, 0 };

   DqRtVmapSetChannelList(handle, vmapid, 1, DQ_SS0IN, channels, flags,
   5);
   ```

4. Start all devices that have channels configured in the VMAP:

   ```
   DqRtVmapStart(handle, vmapid);
   ```

5. Specify how much input data to transfer during the next refresh.

```
DqRtVmapRqInputDataSz(handle, vmapid, 0, numScans*sizeof(uint16),
&act_size, NULL);
```

6. Synchronize the VMAP with all devices:

   ```
   DqRtVmapRefresh(handle, vmapid);
   ```

7. Retrieve the data acquired by device 0:

   ```
   DqRtVmapGetInputData(handle, vmapid, 0, numScans*sizeof(uint16),
   &data_size, &avl_size, (uint8*)bdata);
   ```

   NOTE: Users can log data in a raw format to save CPU cycles and later scale the data offline on the host PC.
   Refer to `DqConv***` data conversion API for descriptions.

8. Stop the devices and free all resources:

   ```
   DqRtVmapStop(handle, vmapid);
   DqRtVmapClose(handle, vmapid);
   ```

7.1.1.6 AVMAP API

In AVMAP mode, the UEIPAC Intel continuously updates data in buffers.

Each input I/O card is programmed to acquire data from its internal FIFO at a rate paced by its hardware clock.

The user application is notified when a user-programmable amount of data is in the FIFO and ready for retrieval.

The content of all the I/O cards' FIFOs is accessed in one operation.

Supported APIs for using AVMAP mode are DqRtVmap*** and DqRtAXmap***.

The following is a quick tutorial on using the AVMAP API (handling of error codes is omitted):

1. Initialize the VMAP to acquire/generate data at 1kHz:

   ```
   DqRtVmapInit(handle, vmapid, 1000.0);
   ```

2. Add channels from the first input subsystem of device 0 (for analog I/O boards, the last parameter is "1" which represents 1 FIFO holding data for all channels):

   ```
   int channels[] = {0, 1, 2, 3 };
   ```

   ```
   DqRtVmapAddChannel(handle, vmapid, 0, DQ_SS0IN, channels, flags, 1);
   ```

3. Set up scan rate for individual device/subsystem:

   ```
   DqRtVmapSetScanRate(handle, vmapid, 0, DQ_SS0IN, 1000.0);
   ```

4. For analog input boards, add the number of actual channels per I/O board from the first input subsystem of device 0:

   ```
   // in header
   #define CHANNELS 4
   //in main program
   DqRtVmapSetChannelList(handle, vmapid, 0, DQ_SS0IN, channels,
   CHANNELS);
   ```

5. Start all devices that have channels configured in the VMAP;

   ```
   DqRtAXMapStart(handle, vmapid, map_type, map_rate, map_cfg, 0);
   ```

6. Set time slot

   ```
   DqRtAXMapSlotAllocate (handle, TRUE, vmapid, 0);
   ```

7. Enable the AVMAP:

   ```
   DqRtAXMapEnable (handle, TRUE);
   ```

8. Issue software trigger to trigger board:

   ```
   DqCmdSwTrigger(handle, vmapid);
   ```

9. Continue to check for asynchronous packet and read data until done with the following functions.:.

   ```
   // Specify (request) how much input data to transfer during the next
   // refresh.
   DqRtVmapRqInputDataSz(handle, vmapid, 0, numScans*sizeof(uint16),
         &act_size, NULL);
   // Retrieve data acquired by device, this call blocks until there is
   // enough data in the FIFO:
   DqRtVmapRefresh(handle, vmapid, 0);
   // Get retrieved data acquired by device 0:
   DqRtVmapGetInputData(handle, vmapid, 0, numScans*sizeof(uint16),
   &data_size, &avl_size, (uint8*)bdata);
   // convert raw data. See the example program SampleAVMap207.cpp and the
   // PowerDNA API Reference for more information on parsing raw data
   DqAdvRawToScaleValue();
   ```

   NOTE: Users can log data in a raw format to save CPU cycles and later scale the data offline on the host PC.
   Refer to `DqConv***` data conversion API for descriptions.

10. Stop the devices and free all resources:

   ```
   DqRtAXMapEnable(handle, FALSE);
   DqRtVmapStop(handle, vmapid);
   DqRtVmapClose(handle, vmapid);
   ```

7.1.1.7 Asynchronous Event API

The Asynchronous API allows you to get notified in your application when a user-programmable asynchronous event occurs.

The types of events supported varies depending on which type of I/O board you are configuring.

Table 7 lists examples of events supported for different I/O board types.

Table 7. Examples of Supported Asynchronous Events

Type of I/O Board	**Type of Asynchronous Events**[4]	**Examples of Supported Boards**
Digital I/O Boards	Edge detection / Change of state and Periodic Events	DNx-DIO-401/3/4/5/6, DNx-DIO-449
Serial I/O Boards	Timeout conditions, RX or TX done, and pattern detection Events	DNx-SL-501, DNx-SL-508
CAN-Bus Boards	FIFO watermark and bus error Events	DNx-CAN-503
Counter/Timer Boards	Count complete Events	DNx-CT-601, DNx-CT-602
Avionics Protocol Boards	Many protocol-specific Events	DNx-1553-553, DNx-429-516

The following is a quick tutorial on using the event API for a digital I/O board, located as device 0 in the IOM and programmed to read the states of all digital input pins on a rising or falling change of state (COS) on port 0 pin 0 (handling of error codes is omitted):

1. Open asynchronous communication with device drivers:
   ```
   DqAddIOMPort(hd, &a_handle, DQ_UDP_DAQ_PORT_ASYNC, 1000);
   ```
2. Set up channels as inputs or outputs
   (this example uses all channels as inputs, which is the default; otherwise, configure I/O directionality: for example, for a DIO-403 use `DqAdv403SetIo()`).
3. Initialize events of device 0 by clearing buffer:
   ```
   DqAdv403ConfigEvents(a_handle, 0, EV403_CLEAR, 0, 0);
   ```
4. Configure a COS event on a rising or falling edge on port 0 pin 0 for device 0:
   ```
   uint8 pos_edge_masks[DQ_DIO403_PORTS] = { 0, 0, 0, 0, 0, 0 };

   uint8 neg_edge_masks[DQ_DIO403_PORTS] = { 0, 0, 0, 0, 0, 0 };

   pos_edge_masks[0] = 1 << 0;
   ```

[4] Refer to SampleAsyncXXX example code for descriptions of types of events supported and specific usage.

```
neg_edge_masks[0] = 1 << 0;
```

```
DqAdv403ConfigEvents(a_handle, 0, EV403_DI_CHANGE, pos_edge_masks,
neg_edge_masks);
```

NOTE: To configure periodic events, use the `DqAdv403ConfigEvents32()` API. Also note that each I/O board type has its own `DqAdvXXXConfigEvents()` API that is used for event configuration.

5. Enable events on device 0:
   ```
   DqRtAsyncEnableEvents(a_handle, 0, 1);
   ```
6. Wait for the next event on device 0. If no event occurs after 1 second, the function returns the error code, “`DQ_TIMEOUT_ERROR`”:
   ```
   ret = DqCmdReceiveEvent(a_handle, 0, 1000000, &pEvent, &size);
   ```
7. Process the event:
   The `DqCmdReceiveEvent()` API returns a `pEvent` structure of size `size`.
   This structure will vary with each I/O board type.
   Refer to `DqAdvXXXConfigEvents()` API for descriptions of the members of correlating `pEvent` structures.

   Note that the returned `pEvent` will always contain the event that was received (`pEvent->event`) and any data associated with the event (`pEvent->data[]`).

   The data returned (`pEvent->data[]`) may need `ntohl` conversion.
   UEI provides byte order conversion API that detect and convert for the correct endianness. For example, use UEI `DqNtohl()` for the conversion of the timestamp:

   ```
   pEv403 = (pEV403_ID)pEvent->data;
   ```

   ```
   tstamp = DqNtohl(hd, pEv403->tstamp);
   ```
8. To finish, disable asynchronous events on device 0:
   ```
   DqRtAsyncEnableEvents(hd, 0, 0);
   ```
9. Close asynchronous communication:
   ```
   DqCloseIOM(a_handle);
   ```

7.1.1.8 Unsupported APIs

APIs that are not mentioned above are not supported on the UEIPAC Intel.

This includes all the ACB (DqACB***), non-RT DMAP (legacy DMAP: DqDmap***), and MSG (DqMsg***) APIs.

8 Installation and Upgrade (Windows Systems)

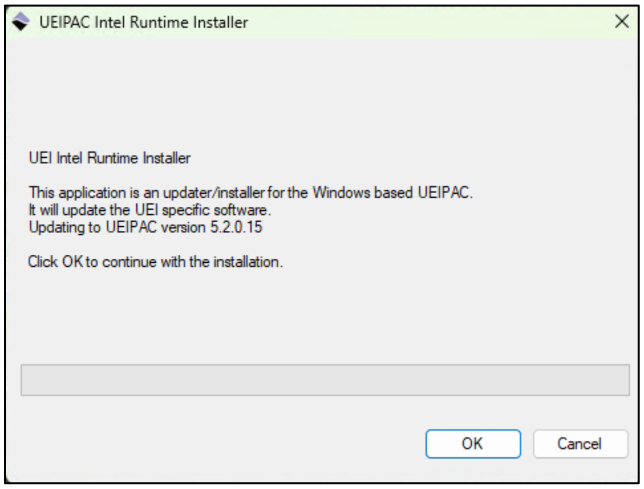
For Windows users, the UEIPAC Runtime Installer comes preinstalled. Updated versions of the installer will allow users to update drivers, runtime libraries, and applications, (e.g., PowerDNA Server) on Windows UEIPAC Intel systems.

To update a UEIPAC Intel Windows system, perform the following steps:

1. Download the UEIPAC Runtime Installer from www.ueidaq.com/downloads and run the installer:

   ```
   UEIPAC_Win_Runtime_Installer_5.x.x.x.exe
   ```

2. The UEIPAC Runtime Installer screen will be displayed as shown in Figure 5. Wait for the installation to complete and press "OK".



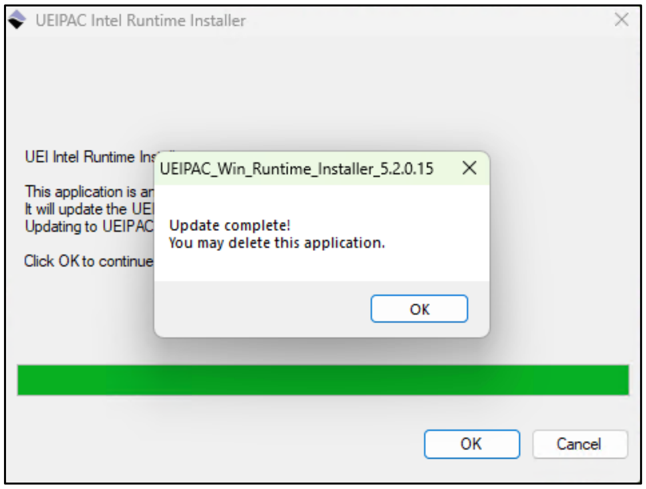
UEI Intel Runtime Installer

This application is an updater/installer for the Windows based UEIPAC.
It will update the UEI specific software.
Updating to UEIPAC version 5.2.0.15

Click OK to continue with the installation.

Figure 5. UEIPAC Runtime Installer

3. When the confirmation screen is displayed (Figure 6), press “OK”.



UEIPAC Intel Runtime Installer

UEI Intel Runtime Ins

This application is ar
It will update the UEI
Updating to UEIPAC

Click OK to continue

UEIPAC_Win_Runtime_Installer_5.2.0.15

Update complete!
You may delete this application.

Figure 6. Confirmation Screen