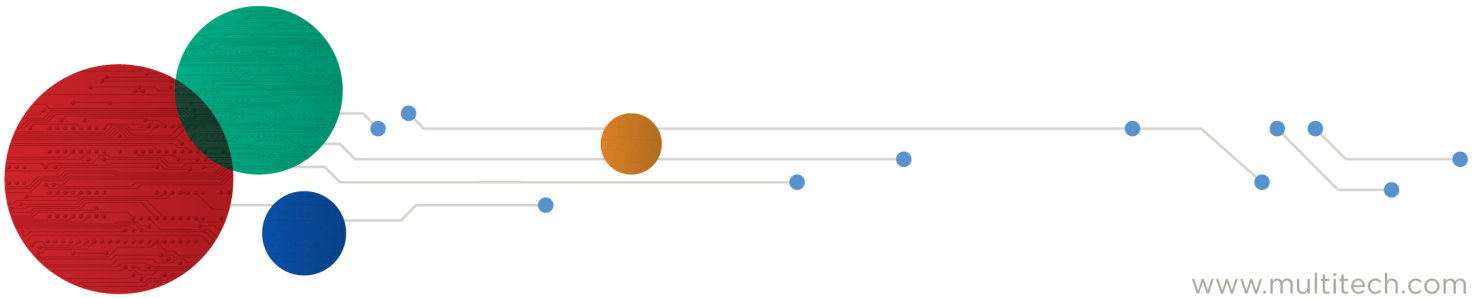
MULTITECH®

DeviceHQ®

App Developer Guide



DeviceHQ App Developer Guide

Part Number: S000756 Rev 1.1

Copyright

Trademarks and Registered Trademarks

Legal Notices

Contacting MultiTech

Knowledge Base

The Knowledge Base provides immediate access to support information and resolutions for all MultiTech products. Visit http://www.multitech.com/kb.go.

Support Portal

To create an account and submit a support case directly to our technical support team, visit: https://support.multitech.com.

Support

Business Hours: M-F, 8am to 5pm CT

Country	By Email	By Phone
Europe, Middle East, Africa:	support@multitech.co.uk	+(44) 118 959 7774
U.S., Canada, all others:	support@multitech.com	(800) 972-2439 or (763) 717-5863

Warranty

To read the warranty statement for your product, visit https://www.multitech.com/legal/warranty. For other warranty options, visit www.multitech.com/es.go.

World Headquarters

Multi-Tech Systems, Inc.

2205 Woodale Drive, Mounds View, MN 55112

Phone: (800) 328-9717 or (763) 785-3500

Fax (763) 785-9874

Contents

Chapter 1 – Overview

DeviceHQ® is a cloud-based tool set for managing the latest generation of MultiTech devices. It provides device management tools such as remote monitoring, upgrades, and configuration. DeviceHQ takes remote device management and maintenance to a new level, by providing an application marketplace, allowing users to browse applications or build their own then easily deploy them to and customize them for remote devices from anywhere.

DeviceHQ supports custom apps. This document includes information on creating and deploying custom apps.

Note: Support for Node-RED/Node.js on MultiTech AT91SAM9G25-based products has been discontinued starting with mPower 5.3.

Requirements

To develop apps, you need:

- A Conduit product (MTCDT, MTCDTIP, MTCAP, MTCAP2), which is configured to be connected to DeviceHQ.

Chapter 2 – Custom Apps

Development Tools And Tips

Downloads

- Custom App SDK (includes custom app template and utility scripts)

Utility Scripts

- **copy-to-conduit.sh** takes a directory and ip address then copies the entire directory to **/media/card/** on the Conduit. This can be used if you want to try manually installing and running your app without **app-manager**.
- **install-to-conduit.sh** takes a directory and ip address. It tars up the app, copies it to **/media/card/** on the Conduit, then uses **app-manager** to manually install the app (bypassing DeviceHQ).

Create the Tarball Package

```
$ cd <your_top_level_app_directory>
$ tar --hard-dereference -hczf <path_to_create_tgz>.tgz *
```

Manual Installation

1. Copy the tarball package to **/media/card/** (if the archive is in the root file system, the installation will fail)

   ```
   $ scp <your_tarball> admin@<your_conduit_ip>:/media/card/
   ```

2. Install the app:

   ```
   $ app-manager --command local --apptype CUSTOM --
   appname <your_app_name> --appfile /media/card/<your_tarball>
   ```

Note: You must uninstall a manually installed app before installing the same app through DeviceHQ.

- Via the Conduit UI's **Apps** page
- Via the command line:

```
$ app-manager --command remove --appid LOCAL
```

Starting and Stopping

- Via the Conduit UI's **Apps** page
- Via the command line:

```
$ app-manager --command start --appid LOCAL
$ app-manager --command restart --appid LOCAL
$ app-manager --command stop --appid LOCAL
```

Check App Status

- Via the Conduit UI's **Apps** page
- Via the command line:

```
$ app-manager --command status
```

Creating a Custom App

The AEP firmware version 1.3 or later supports installing and running user-created custom applications as well as managing them from DeviceHQ. You can disable Node-RED to save system resources.

Custom Apps Requirements

- Use a supported language: **C/C++, Python, Nodejs**
- The app can be installed either on an SD card or in flash.

  You can use an optional boolean variable to manage this or otherwise the app-manager takes care of it automatically based on the presence of an SD card. For more information, refer to manifest.json Details.
- Application is **packaged as a gzipped tar file** containing all required files.
- Use the new **app-manager** program on the Conduit to manage custom apps

  This is exclusively used to install, uninstall, start, and stop custom apps.

Required Application Structure

All required files must be present for successful installation.

Creating A Custom Application

To create a custom app:

1. Download the custom app SDK template and extract the enclosed template tarball into your application directory.
2. Modify **manifest.json** to contain the information about your app. For more information, refer to manifest.json Details.
3. Set up app dependencies:

a. If your app doesn't have dependencies to install, remove the two sample IPK entries in **provisioning/p_manifest.json**

b. If your app does have dependencies to install:

1. Create IPK files of all dependencies and place them in the **provisioning** directory
2. Update **provisioning/p_manifest.json** with all your IPK dependency files.

For details, refer to p_manifest.json Details.

3. Modify the **Install** script if you need extra dependency installation or other install setup beyond installing IPK files. This script is used to install and uninstall your app. For details, refer to Install Script.
4. Modify the **Start** script to start and stop your application process(es) as needed. For details, refer to Start Script.
5. If you have config files you want to update through DeviceHQ, place them in the **config** directory.
6. Modify **status.json** with the default AppInfo string of your choosing. For details, refer to status.json Details.
7. Organize the rest of your application content within this directory any way you like. The Conduit only cares about the location of the required and optional files noted above.
8. Create a gzipped tar file (tarball) of this directory, ensuring all required files are present and correct. The tarball doesn't need to match the app name in manifest.json. The first level application content like manifest.json must be at the top level of the tarball just like the custom app template. To create a tarball on Linux:

```
$ cd <your_top_level_app_directory>
$ tar --hard-dereference -hczf <path_to_create_tgz>.tgz *
```

manifest.json Details

- Required at the top level of a custom app tarball
- The file must be JSON format.
- After installation, it will be located at **/media/card/<app_name>/manifest.json** or **var/config/app/<app-anme>/manifest.json.**
- Can place an optional boolean variable into manifest.json that determines where to save the app: 1) to an SD card, or 2) in flash. Otherwise this is handled automatically (see details below).

Format Details

Key	Value Type	Description
AppName	String	Application Name, used for the installed app directory name and displaying in the UI and on DeviceHQ.
AppVersion	String	Application version, DeviceHQ uses this to distinguish between versions.
AppDescription	String	Application purpose, what does it do.
AppVersionNotes	String	Any applicable notes for the particular version of the application displayed on DeviceHQ.

Key	Value Type	Description
SDCard	Boolean	Optional variable determines where to save app. If true, it saves to SD card. If false, saves in flash.

Example manifest.json

```
{
    “AppName” : “LoraSensor”,
    “AppVersion : “1.0”,
    “AppDescription” : “Example app”,
    “AppVersionNotes” : “Added new feature”
}
```

Optional SDCard Boolean Variable

You can use an optional boolean variable, **SDCard**, to determine where to save the app. If **SDCard** is true and an SD card is present, the app saves to the SD card (in /media/card/<app_name>). If false, the app saves in flash (in /var/config/app/<app_name>).

NOTE: If **SDCard** is true but the card is not present, the installation fails with an error.

Here is an example of manifest.json using **SDCard**:

```
{
“AppName” : “<string_value>”,
“AppVersion : “<string_value>”,
“AppDescription” : “<string_value>”,
“AppVersionNotes” : “<string_value>”,
“SDCard” : <boolean>
}
```

If you do not use **SDCard** in the file, the **app-manager** determines if the SD Card is inserted automatically and installs in flash or to the card for you.

SDCard Algorithm

The following is pseudo code logic for handling the installation of a Custom Application with these enhancements:

```
if “SDCard” present in manifest.json:
     if “SDCard” == true:
          if SD Card is inserted:
               Install in /media/card/<app_name>
          else:
               FAIL
     else:
          Install in /var/config/app/<app_name>
else if SD Card is inserted:
     Install in /media/card/<app_name>
else:
     Install in /var/config/app/<app_name>
```

p_manifest.json Details

p_manifest.json contains a list of IPK dependencies to be installed by the Install Script.

- Optionally present in the provisioning/ directory.
- Must be JSON format with a pkgs array.
- This file is exclusively used by the Install Script to install the listed dependencies.
- After app installation, this file is located at /media/card/<app_name>/provisioning/p_manifest.json. t list item.

Format Details

Keys	Value Type	Description
pkgs	Array	
FileName	String	Package file name
type	String	Only "ipk" is supported at this time.
pkg_name	String	Actual package name (field in ipk control file)

Example p_manifest.json

```
{
  pkgs: [
    { "FileName": "file1.ipk", "type": "ipk", "PkgName": "name1" },
    { "FileName": "file2.ipk", "type": "ipk", "PkgName": "name2" }
  ]
}
```

Warning for Updating Base Packages

The default **Install** script will both **install** and **remove** any opkg dependencies in p_manifest.json using opkg with the **–force-depends** argument. If you are updating a system library removing a package in this manner may have unintended consequences. If you are upgrading these packages, the opkg commands defined by **OPKG_CMD_PREFIX** and **OPKG_CMD_PREFIX_R** in "Install" should be modified to remove **–force-depends**.

Install Script

Install Script Details

A template **Install** script is provided in the custom app template tarball. The **Install** script is responsible for installing any app dependencies.

- Required to exist at the top level of the application tarball
- It must accept the following command line arguments: **install**, **postinstall**, and **remove**.
- The provided template is configured to install IPK files listed in p_manifest.json, but this script can be fully customized to your needs including removing the IPK installing code.
- This script is the place to download files from the internet or perform actions like **npm install**.

- The **install** and **postinstall** functions must be written in such a way that they can be successfully executed multiple times.
- This script is always run as **root**. Don't forget to make the permissions on the Install script executable by the appropriate user.

The below process flows show exactly when and how the **Install** script is executed.

Install Process

1. **app-manager** launched to install a custom app either from a DeviceHQ command or manually from the command line.
2. If a version of the app is already installed, **app-manager** stops it via theStart script.
3. **app-manager** either downloads the app tarball from DeviceHQ or if manually installing, checks it's existence at the given path.
4. **app-manager** creates a directory in **/media/card/** with the app name from manifest.json.
5. **app-manager** extracts the contents of the app tarball into that directory.
6. If a config file was selected when queuing the Install command from DeviceHQ, **app-manager** downloads the config file and places it in the **config/** directory, overwriting any existing files.
7. **app-manager** executes the **Install** script with the **install** argument.
8. **app-manager** saves the meta data for the newly installed app.
9. **app-manager** executes the **Install** script with the **postinstall** argument.
10. **app-manager** starts the app via the Start script.

The Conduit checks into DeviceHQ to report new Installed Apps information.

Uninstall Process

1. **app-manager** launched to uninstall a custom app.
2. **app-manager** stops the app via the Start script.
3. **app-manager** executes the **Install** script with the **remove** argument.
4. **app-manager** deletes the entire app directory from **/media/card/**.

The Conduit checks into DeviceHQ to report new Installed Apps information.

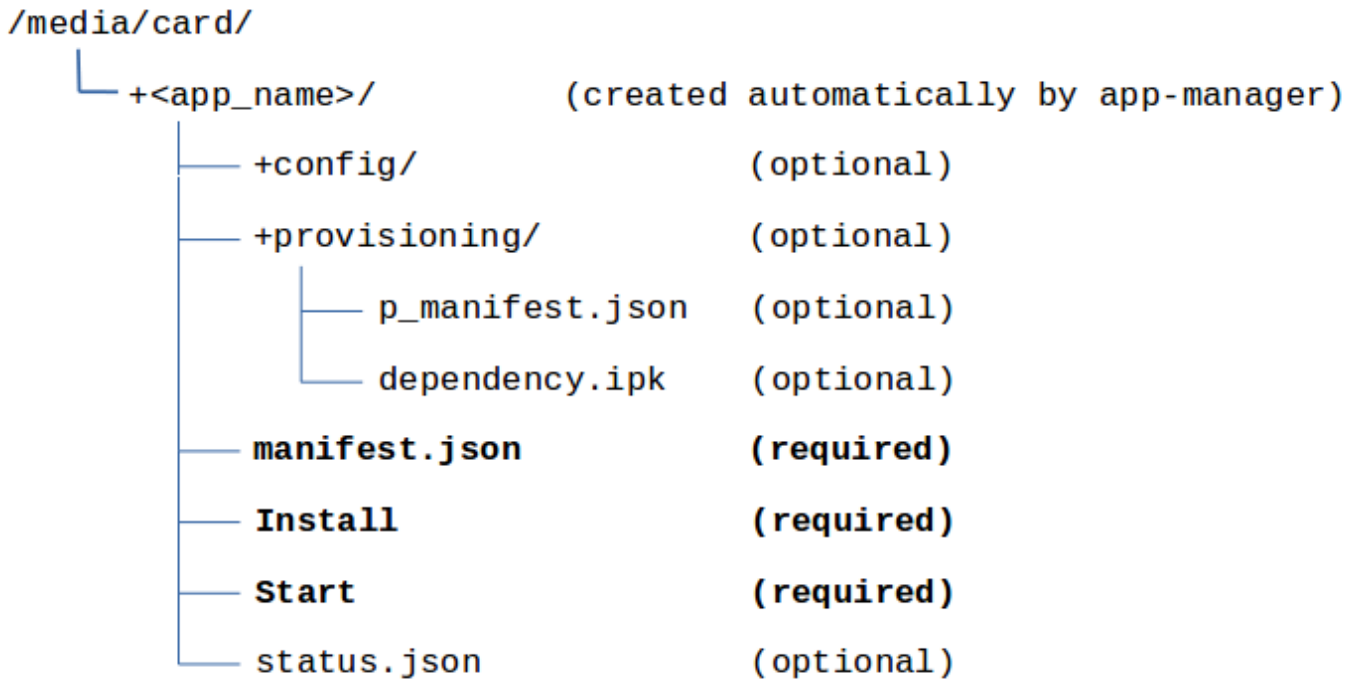
Conduit Firmware Upgrade Process

1. Conduit firmware upgrade initiated.
2. Conduit reboots and upgrades firmware.
3. On first boot after upgrade, the **/etc/init.d/customapp** script does the following for each installed app:
    a. Launches **app-manager** to install dependencies,
        i. **app-manager** executes the **Install** script with the **install** argument,
        ii. **app-manager** executes the **Install** script with the **postinstall** argument,
    b. Launches **app-manager** to start the app,
        i. **app-manager** starts the app via the Start script.

Warning for Updating Base Packages

The default **Install** script will both **install** and **remove** any opkg dependencies in p_manifest.json using opkg with the **–force-depends** argument. If you are updating a system library removing a package in this manner may have unintended consequences. If you are upgrading these packages, the opkg commands defined by **OPKG_CMD_PREFIX** and **OPKG_CMD_PREFIX_R** should be modified to remove **–force-depends**.

Installed Application Structure



Start Script

Start Script Details

The custom app template includes a **Start** script template. The Start script is responsible for starting and stopping the application.

- Required to exist at the top level of the application tarball
- Must accept the following command line arguments: start, stop, restart, and reload.
- The provided template shows how to use start-stop-daemon to start and stop your app, but this script can be fully customized to use whatever start/stop mechanism you wish.
- Apps will be automatically started after install and on boot and stopped on shutdown.
- This script is always run as root. Don't forget to make the permissions on the Install script executable by the appropriate user.

The Install process flows show exactly when and how the Start script is executed.

Install Process

1. **app-manager** launched to install a custom app either from a DeviceHQ command or manually from the command line.
2. If a version of the app is already installed, **app-manager** executes the Start script with the stop argument.
3. **app-manager**installs the app via the Install script.
4. **app-manager**r executes the Start script with the start argument.
5. The Conduit device checks into DeviceHQ to report new Installed Apps information.

Uninstall Process

1. **app-manager** launched to uninstall a custom app.
2. **app-manager** executes the Start script with the stop argument.
3. **app-manager** uninstalls the app via the Install script.
4. The Conduit device checks into DeviceHQ to report new Installed Apps information.

Boot Process

1. Conduit starts boot process
2. The **/etc/init.d/customapp** init script is executed with priority 95
3. **customapp** queries **app-manager** for all installed apps
4. For each app, **customapp** launches **app-manager** to start it
5. **app-manager** executes the **Start** script with the **start --initd** arguments

Shutdown Process

1. Conduit shutdown initiated
2. The **/etc/init.d/customapp** init script is executed with priority 95
3. **customapp** queries **app-manager** for all installed apps
4. For each app, **customapp** launches **app-manager** to stop it
5. **app-manager** executes the **Start** script with the **stop --initd** arguments

Config Install Process

1. **app-manager** launched to install an app config from a scheduled DeviceHQ action
2. **app-manager** downloads the new config file and places it in the **config/** directory, overwriting any existing files.
3. **app-manager** executes the **Start** script with the **reload** argument

status.json Details

By default, the basic status of your custom application will be displayed in the Conduit® UI and on DeviceHQ® and will generally be **Started** or **Stopped**. By default, we don't know whether your application is truly running or not.

However, if you provide **status.json** in the format described below, app-manger check if a process with the given **pid** is running. If it is, your app status will be reported as **Running**. If it was previously running and app-manager discovers it's no longer running, your app status will be **Failed**.

Additionally, you can set the **AppInfo** field of **status.json** to use as a custom status reporting mechanism. The **Status** field in the UI and on DeviceHQ will always be set by app-manger, but the **Info** field will be read directly from **status.json** if it's available.

status.json requirements:

- Optionally present at the top level of a custom app tarball
- The file must be JSON format.
- For **app-manager** to use this file after installation, it must be located at **/media/card/<app_name>/status.json**

Format Details

Key	Value Type	Description
pid	Integer	The PID of your app process. **Your app must set this at runtime if you want full status reporting.** Each supported language has a built in mechanism for getting your current PID. See the provided sample apps for examples. `app-manager` will use this PID to determine app status by doing `kill -0 <pid>` to see if the process is running. See below information on possible app statuses.If you include this file as part of the tarball, it's recommend to set the pid to 0 as default.
AppInfo	String	Can contain any string up to 160 characters and will be displayed on the Conduit and DeviceHQ UI's. This is a good way to display application state or errors.

Example

```
{
   "pid": 2374,
   "AppInfo": "Started processing"
}
```

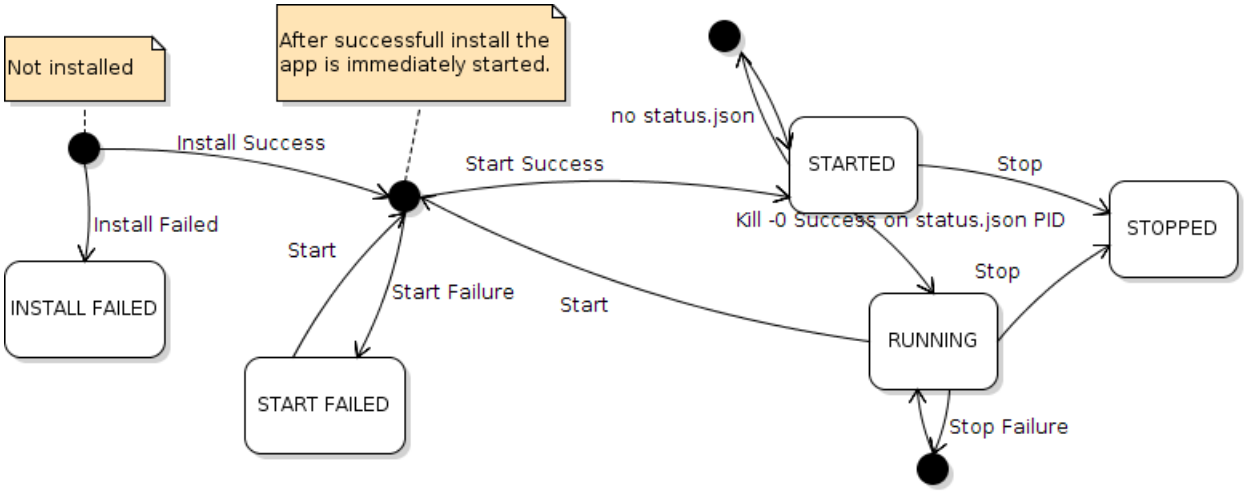
Status Values

Your app **status** as determined by app-manager will be displayed in the Conduit and DeviceHQ UI's. This value is determined by whether a process with the **pid** as set in **status.json** is running or not.

Possible statuses:

1. STARTED
2. RUNNING
3. STOPPED
4. FAILED
5. INSTALL FAILED
6. START FAILED

Possible status transitions:



Example Apps

The following TAR files are example applications.

- Nodejs app with web view: express-hello-world v1.4
- C++ app with boost dependency: boost-query-api v1.0.2
- Python gpio toggle (requires MTAC-GPIO accessory card): gpio-led v1.3

Using Monit to Monitor Custom Apps

You can also use monit to monitor your custom applications. For information on installing the monit package and its use, refer to the monit page on the MultiTech Developer Resources site.

Backup Custom Apps

You can backup/save Custom Apps when you save them in User Defined Defaults of the device. You must install them in the /var/config directory using the User Defined Defaults feature.

Chapter 3 – Node-RED Apps

Note: Support for Node-RED/Node.js on MultiTech AT91SAM9G25-based products has been discontinued starting with mPower 5.3.

Node-RED is a development tool for connecting hardware and online services.

There are three ways to build a Node-RED App:

- Create a new application by dragging nodes onto the workspace and connecting them to create a flow.
- Update an application by importing an existing Node-RED flow via the Import Menu.
- Download an existing application and then upload the changes as a new application.

This chapter provides information on creating apps using a Conduit product and Node-RED, but detailed help for Node-RED is beyond the scope of this document. For more information about developing Node-RED apps, refer to the Node-RED site at https://nodered.org/. A good tutorial for developing Node-RED apps can be found at: https://www.youtube.com/watch?v=f5o4tIz2Zzc.

For help with Conduit products, refer to the mPower Conduit AEP Software Guide.

Developing Node-RED Applications

To get started developing a Node-RED app:

1. In a browser, connect to the Conduit's local UI at https://192.168.2.1.
2. Select **Apps** from the menu on the left.

   This page displays apps installed on the Conduit. If no apps have been installed, the only app that is shown is the Development app. The Development app is the default workspace. Use this to begin developing a new app.

3. Click **Launch Node-RED**.
4. Log into Node-RED.

   The workspace of the current running app is displayed. If there are no apps running, the workspace is blank.

Creating a New Node-RED Application

The following a basic overview for creating a Node-RED application.

For more detailed information on creating a basic flow, go to: http://nodered.org/docs/getting-started/first-flow.html.

For more detailed information on creating a more advanced flow, go to: http://nodered.org/docs/getting-started/second-flow.html.

1. Drag and drop any of the pre-defined nodes onto the workspace.
2. Connect the pre-defined nodes together by dragging between the output port of one to the input port of the other.
3. To see the app run, click **Deploy**. Results appear on the **debug** tab. Whenever you change a flow, click Deploy to see the new behavior.

Once the app has been created, tested, and deployed, you can upload it.

Create a New Application from an Existing Application

You can download an app from DeviceHQ and use it as a base for a new application. To do this, download the app that you want to your Conduit. This app will serve as the base for your new app.

1. Log into the Conduit.
2. Click the **Apps** link in the menu on the left.
3. Click **Installed Apps** to see the apps currently installed on the Conduit. Make sure the app you want to build from is currently running.
4. Open Node-RED.
5. Make desired changes to the base app. To test the changes and see the app run, click the **Deploy** button.

   To save any changes you make to the flow, you must click the **Deploy** button.
6. Once the app has been updated and tested, upload the new app. For information on how to upload an app, see Uploading a Node-RED Application to DeviceHQ through a Conduit.

Chapter 4 – Working with Apps

Uploading a Node-RED Application to DeviceHQ through a Conduit

To upload a Nod-RED application from the Conduit interface:

1. Select **Apps** from the menu bar on the left.
2. Click **Upload** .
3. Enter the **application name**.
4. Enter the **version.**
5. Enter your DeviceHQ login credentials including email and password. Note: For security purposes, the credentials are not saved on the device. For more information, refer to the mPower AEP Software Guide
6. Click **Upload**.

Accessing the DeviceHQ Developer Page

To gain access to the Developer page for uploading apps to the Store, your account needs a Developer Key. If Developer appears in the top navigation bar, you have already have a developer key.

To generate a Developer Key in Device:

1. If not logged in to DeviceHQ, log in.
2. Click your user name in the upper right corner of the page.
3. Click **My Profile.**
4. Click **Generate Developer Key.**

Developer is now an option in the top navigation bar.

Adding an Application to the App Store

To add an app to the Store:

1. If not logged into DeviceHQ, log in.
2. Click the **Developer** tab.
3. Click **Upload App.**
4. Click **Choose File** and select the app file you want to add to the store.
5. On the App list, find the newly uploaded app and click the **Edit** icon.
6. *For Node-RED apps*, click the **Versions** tab and click the Edit icon to edit the version. Note: For custom apps, Version allows you to download or delete the app only
7. *For custom apps only*, select the app **Configuration** file to upload or select an application configuration file. For a new configuration file, click **Upload Configuration**, **Choose File**, enter a **Description**, and click **OK**.
8. Change the status to **Active** and enter a short version description,
9. Click **Save**.
10. On the **Description** tab:
    a. If desired, change the name of the app.
    b. Enter a description of the app. This appears in the App Store.

c. Choose an icon for the app.

11. On the **Publish** tab, select the app Status. Options are Public, Private, and Inactive.
    - All DeviceHQ users will be able to see and use **Public** apps.
    - Only users of the same account will be able to use **Private** apps.
    - **Inactive** apps do not appear in the store.
12. Select the **License Agreement** you want for your app. Options are GNU General Public License, MIT License, and Custom. If you select custom, you'll be prompted to enter a TOS URL or TOS text.
13. Click **Save**.

The application appears in the store.

Updating an Existing Application

The following process is used to update an existing application, it explains how to edit an application and upload the changes to DeviceHQ. This process assumes that you have downloaded an application to your Conduit.

1. Log into the Conduit's UI.
2. On the menu on the left, click **Apps**.
3. Click **Installed Apps**.
4. Verify that the app to be modified is currently running.

   If the app is not currently running, click the **pencil** icon in the installed apps list and it will run.

5. Open Node-RED.
6. Update the app.
7. Click **Deploy** and verify that app has been updated.
8. Return to the Conduit UI and click **Apps** on the menu on the left.
9. Click **Upload App**.

   a. Enter the username and password for your DeviceHQ account.
   b. Choose **Existing App** from the Type drop down menu.
   c. The **App ID** will be filled in automatically.
   d. Enter a **Version**. The old version number is displayed by default.
   e. Click **Publish**.

   A message will display at the start of the upload and once the upload is completed.

Configuring an Updated Application

Once an app has been updated, the new version must be configured so it will show in the Store. To configure a new version:

1. Log into DeviceHQ.
2. Click the **Developer** button on the top menu. You app will be listed.
3. Click the **Edit** icon for the updated app.
4. Click the **Versions** tab.
5. Click the **Edit** icon for the new version.

a. **Type** is currently restricted to apps only.
b. Set the status to **Active**. The previous Active version will be set to Archived.
c. The **Notes** field is used to track version changes.
d. Click the **Save** button. The new version information will be displayed.

6. Click the **Close** button. The app now appears in the Store with the updated Active version.