Requirements Document

Programmable Asset Tracker

Updated Version: **1.8.** Date: **Oct-21-2022**

# 

[**Solution Architecture**](#_i6zayvcxr7k) **3**

[**Project Timeline & Deliverables**](#_fry0d3zcgkca) **4**

[Proposed Timeline\*](#_1l2bcwa170s) 4

[**High-Level Product Customer/User Information (Informative)**](#_bub8ghpg0rzu) **5**

[Basic User Story (in Production Deployments)](#_mrs2j3z6txdp) 8

[**Hardware Requirements**](#_yn5gai7ga4ma) **9**

[PoC Phase Hardware](#_u4ajw7ai7t7l) 9

[Field Trial/Production Hardware](#_v1fnd3642yes) 10

[**Device Software Requirements**](#_bihm3w1suui) **11**

[General software requirements](#_gzb31azdeqjn) 11

[Quality Metrics](#_el0sggjvpul0) 12

[Application Functionality](#_ey85nr5bvpu4) 13

[Device Evaluation UI (to be finalized)](#_9vlz81lkzivy) 19

[Library functionality](#_oirtzrp8qkbx) 19

[Out of Scope](#_dzgoy8v59zrs) 19

[**Cloud Agent Software**](#_tspfrs452v9k) **20**

[Simplified Proposal for Cloud Integration](#_rc0fa7kp7osj) 20

[**Device Manufacturing & Test Software Requirements**](#_5roub7rr7fzg) **22**

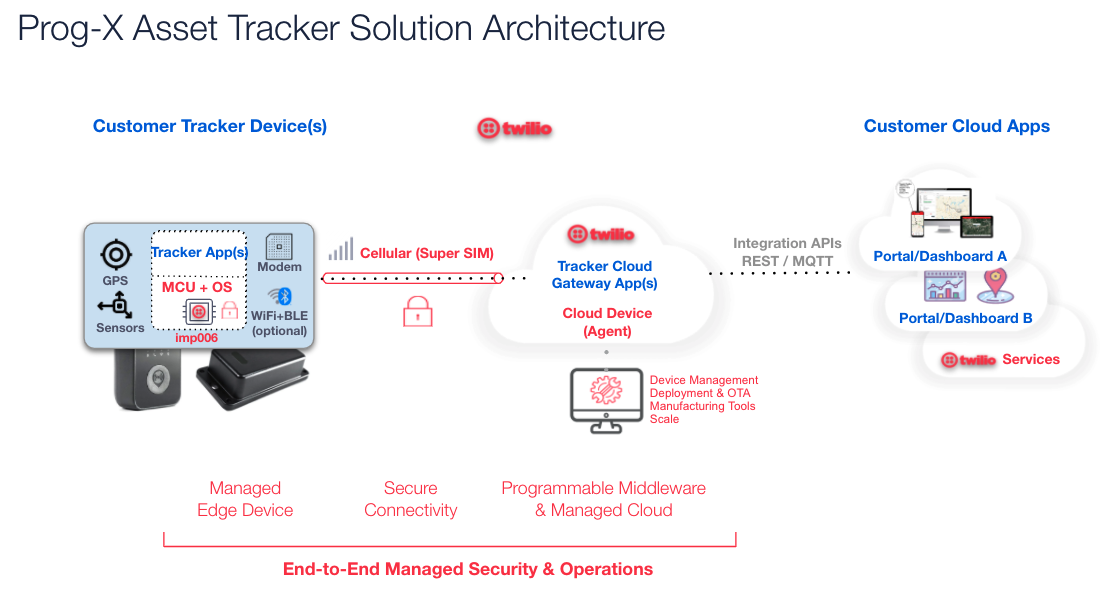
[**Appendix: Planned Testing Checking by Customer**](#_q729zzt6g6t4) **22**

[**Addendum: Field Trial 2 Change Requests**](#_dmscusq7ba3v) **23**

# Solution Architecture

The Twilio Electric Imp platform allows to readily define a solution architecture with the following components:

* Tracker device with sensors and communication interfaces, based on imp006 module with impOS
* Updatable tracker device application(s)
* Global cellular cloud connectivity via Twilio Super SIM
* Twilio Electric Imp managed cloud and programmable middleware (cloud agent)
* Updatable cloud agent (gateway application) with integration one or more cloud tracking portals/dashboards
* Tracker portals/dashboards (outside of project scope)



# 

# Project Timeline & Deliverables

## Proposed Timeline\*

|  |  |
| --- | --- |
| Sep-Oct 21 | * Agreement on engagement * Initial requirements definition * Non-binding commitment based on success criteria * Work on PoC readiness starts * Scoping & preparation |
| Nov 21 | PoC development:   * Design & development work |
| Dec 21 | PoC phase [“Will it work”]:   * PoC hardware (based on imp006 breakout) * PoC device software and simplified cloud integration (joint) * PoC testing by Lead Customer (doesn’t include *HW* durability, approx. 2 weeks duration) |
| Jan 22 | Field trial development:   * Finalized requirements * Design & development work * Final simplified cloud integration (joint) * Pre-production hardware design & prototypes |
| Feb 22 | Field Trial phase [“Does it Work”]:   * Initial device software and cloud integration blueprints * If necessary, blueprint customization for Lead Customer * Device manufacturing & testing software blueprint * Pilot production run with selected ODM * Field Trial testing by Lead Customer   + Field trial size: Approx. 100 units   + We ask Lead Customer purchase field trial units at cost   + Approx. 4-6 weeks duration   + Note: With 100 units, this may already count as pilot rollout |
| Apr 22 | Contract and Production Pilot phase:   * Final production hardware design + certifications |
| Jun 22 | Production Ramp and Rollout:   * Production run with Lead Customer CM * Sale and rollout through Lead Customer |

\* Production schedule is dependent on hardware availability/lead times and ODM schedule

# High-Level Product Customer/User Information (Informative)

## Target Audiences and Customer Journey

This section is a high-level description of the audiences envisioned for the tracker to identify their needs and how they interact with the Programmable Asset Tracker. While the focus is on production customers (and the business they represent) it is important to realize that most customers start with evaluations and so the journey should be smooth so that a high percentage of evaluation customers progress to production.

The remainder of this document assumes the following aspects of the initial provisioning flow have already happened (defined outside of this doc):

1. The customer has a Twilio account with the tracker Super SIM(s) registered and activated
2. The user has an Electric Imp account with a predefined device group provisioned with
   1. Programmable Asset Tracker application
   2. Programmable Asset Tracker environment variables default settings. This typically includes a pre-created temporary user-unique Ublox assist token (Note: We do not pre-create a Google Maps token, this would only exist if the user has created and entered one by themselves).
3. For multi-pack tracker purchases:
   1. The Electric Imp account is promoted to production eval
   2. Trackers are blessed to the account and assigned into the predefined device group
4. For single tracker purchases:
   1. The Electric Imp account remains a development account
   2. Trackers are unblessed, not assigned to the customer’s account

A. Initial evaluation customer

This is a customer who is interested in the concept of the Programmable Asset Tracker and buys a single unit in order to do an initial evaluation and see if the tracker may suit their needs. The main goal of this customer is to quickly get the tracker running and understand the key functionality and data it provides (location, alerts, etc).

This exercise must be easy and should require as little setup or technical knowledge as possible. To optimize the out-of-the-box experience this evaluation should have no or only minimal dependencies and further steps to perform.

Key expectations:

* Experience optimized for single tracker
* User has basic understanding of asset tracking concepts, but no deep technical skills
* Works out-of-the-box with minimal set-up
* Simple operation/evaluation via device (agent)-provided web UI
* Note: May have reduced battery life due to evaluation settings (frequent reporting, etc)

User Story

1. User purchases a single tracking device from the Twilio Console (as per initial provisioning flow, this is an unblessed developer device)
2. User installs Electric Imp BlinkUp (developer) app and logs into their existing account
3. User removes ‘tamper detection patch’ to activate tracker (exits ‘shipping mode’)
4. User opens the tracker and uses the BlinkUp app to blink it up and assign it to the predefined device group
5. Tracker starts application with the defined configuration in the environment variables
6. BlinkUp app detects the web UI of the device (agent) and offers to open it in a browser
7. User views tracker data such as location and sensor information in the web UI, experiments with the tracker (walks around, shakes it, mounts it to an asset, etc)
8. **Optional configuration:** User can make changes to some configurations (such as reporting interval) in web UI to modify the operation
9. After successful evaluation user moves on to extended evaluation

B. Extended evaluation/field trial customer

Customers who have successfully completed the initial evaluation will likely want to do more extensive evaluation, for example buying 5-10 more trackers, configuring them, and then using them to track multiple assets for a few weeks in a field trial situation. The main goal of this customer is to understand if the trackers would work (or can be customized to work) for their solution requirements.

This exercise should still be easy to set up and perform, but will require some technical skills, including deeper configuration as well as potentially modifying the application code or cloud integration.

Because this scenario involves multiple trackers it is expected the customer will want to configure all trackers in a similar fashion and then aggregate all tracker data into a single cloud service for viewing/dashboarding. To make this easy the Programmable Asset Tracker should include at least one ready-made integration with a 3rd party cloud (tracking) portal. Alternatively, the customer may want to integrate the trackers directly with their own backend services/applications.

Key expectations:

* Experience optimized for 5-10 trackers
* User has at least basic technical skills and understanding of web and device technologies, and optionally may need software development skills to customize the application
* Basic set-up to configure all trackers as well as configure integration with cloud service/portal (further configuration expected via IDE, primarily environment variables)
* Evaluation via pre-integrated 3rd party portal or customer cloud service
* Note: May have reduced battery life due to evaluation settings (frequent reporting, etc)

User Story

1. User purchases 5-10 tracking devices from the Twilio Console (as per initial provisioning flow, these devices are blessed to the customer’s account)
2. User removes ‘tamper detection patch’ to activate trackers (exits ‘shipping mode’)
3. Because the trackers are already assigned to the predefined device group the trackers start the application with the defined configuration in the environment variables
4. **Optional configuration:**
   * User points a browser to the URL of one device (agent) with the web UI
   * User uses web UI to configure that one device, including configuring the cloud service (pre-integrated portal via drop-down or custom URL for other service)
   * User uses web UI to show the configuration as a JSON structure (cfg-json) and copies it, then pastes it into the JSON configuration structure into the device group environment variable field and overriding the previous values
   * User redeploys the application to all trackers so that all devices now have the same (changed) configuration.
5. **Optional modification:** User may modify the application to customize it to their needs, or get support to do so if they don’t have the skills
6. User proceeds with evaluation
7. After successful evaluation user moves on to production

C. Production customer

Customers who have successfully completed the extended evaluation/field trial and have decided in favor of the Programmable Asset Tracker will move to production, with volumes ranging from dozens of units for a small deployment to many thousands or tens of thousands of units for large deployments. The main goal of this customer is to complete the full integration of the tracker into their solution for optimal business results and business and roll out the solution at scale.

The customization needs vary and depend on the requirements of each customer. This can include custom configuration(s), modifying the tracker application, and/or integration into the customer’s product stack, backend systems, data models, or workflows. The Programmable Asset Tracker supports this customization through various mechanisms, all the way to extensive modifications of the tracker application code if necessary. Customization can be done by the customer or by a development/integration partner.

Key expectations:

* Supports production deployments
* User has technical skill to productize the solution (or has partners who can support)
* Includes configuration/customization/modifications and is possible via IDE, environment variables, and/or open source application code and Electric Imp tooling
* Operation via 3rd party portal or customer cloud service

User Story

Punting on the user story for now as this is very variable based on customer needs. Basically, every aspect of customization is possible.

## Basic User Story (in Production Deployments)

This is the high-level user story on how a typical end user (tracking customer) would interact with the product, focused on the device/IoT platform portion of the functionality:

1. Manufacturing / storage / shipping
   * Tracker is tested, batteries inserted, pre-provisioned, and set to ‘shipping mode’ (permanent sleep)
   * Tracker is stored in warehouse, then shipped to customer (user)
2. User installation
   * User unpacks tracker and mounts it on asset
   * User removes ‘tamper detection patch’ to activate tracker (exits ‘shipping mode’) **(not in PoC)**
   * User uses mobile/web app (note: part of customer solution) to enter serial number or scan QR code and associate tracker to account and asset
   * Note: Mechanism to associate tracker with asset is still being investigated - this is not part of project scope.
3. User uses tracking system via mobile/web app, with the following possible operations:
4. Device configuration (based on tracker ID/serial number)
   * User views last configuration values in tracking portal. User is able to force re-reading configuration values from device to ensure the tracking portal reflects the actual configuration state of the device.
   * User sets individual configuration values in tracking portal. Possible configuration values include (list may be incomplete):
     + Sensor reading period
     + Normal reporting period
     + Alerts on/off & thresholds (for tamper detect, sensors, battery, etc)
     + **(not in PoC)** On-device geofence on/off & center/radius
     + Tracking mode on/off & reporting period
     + **(not in PoC)** Repo/theft mode on/off & reporting period/schedule
     + **(not in PoC)** Known BLE beacons and locations
   * User saves configuration state in tracking portal. All changes are queued for the device and marked as pending in tracking portal. When device connects, changes are pushed and changes confirmed as applied by device are marked as confirmed.
5. Normal reporting mode
   * User views tracker data (tracker ID, last connection, position, sensor values, battery level) on tracking portal. Values are updated on tracking portal when tracker reports periodically.
6. Alerts
   * User configures alerts in tracking portal (tamper detect, sensor alerts, battery alerts, geofence alerts, etc)
   * If alerts trigger on device they are shown in tracking portal in semi-real time
7. **(not in PoC)** On-device geofencing
   * User enables on-device geofencing in tracking portal by configuring geofence (center, radius)
   * If device enters/exits geofence, alert is shown in tracking portal
8. Tracking mode
   * User enables tracking mode in tracking portal by configuring tracking (reporting period)
   * If device enters/exits tracking mode, alert is shown in tracking portal along with position updates
9. **(not in PoC)** Repo/theft mode
   * User enables repo/theft mode in tracking portal by configuring repo/theft (schedule)
   * If device enters/exits repo/theft mode, alert is shown in tracking portal along with position updates

# Hardware Requirements

## PoC Phase Hardware

imp006 breakout board

* BG96 (LTE-M1 + 2G fallback), embedded Super SIM (Note: BG96 GNSS used for only for initial PoC)
* Added ublox M8N module incl. antenna, attached via UART
* Temp/humidity sensor (humidity not used)
* Accelerometer
* Light sensor (BlinkUp sensor)
* WiFi/BLE: Passive positioning via WiFi APs (2.4 GHz) and BLE beacons
  + ~~Initially use on-board 1MW~~
  + Added ESP32-WROOM-32 attached via UART, e.g. https://www.mikroe.com/wifi-ble-click
* LEDs
  + BlinkUp status
  + Programmable status LED for user notifications
* No wake button
* Rechargeable 2000 mAh LiPo battery
* Battery level monitoring (initially via voltage level. If not accurate enough, need to consider battery gauge via coulomb counter)
* Antennas: Cellular, GNSS, WiFi/BLE (for passive reading)
* Existing NewAge IP67 enclosure

## Field Trial/Production Hardware

* ~~Based on imp006, BG96 (LTE-M1 + 2G fallback), embedded Super SIM, BG96 GNSS~~
* Based on tracker design, BG95-M3 (LTE-M1 + 2G fallback), embedded Super SIM, (Note: BG95-M3 GNSS not used)
* ublox GNSS module incl. antenna, via UART
* On-board sensors:
  + Using temperature sensor in LIS2DH12 (LISDH12TR) (Note: This is just for basic on-board temperature sensing, so limited accuracy is OK. No HTS221 on device)
  + Accelerometer (LIS2DH12)
  + Light sensor at bottom, connected to imp006 ‘wake’ pin (light/dark detection, to detect tampering/removal of unit. Also to wake from ‘shipping mode’ deep sleep). Plan to use hardware.lightlevel()
* ESP32 module with ESP-AT
  + For passive WiFi scanning and BLE
  + Connection to enable DFU of ESP32 firmware from imp
* WiFi/BLE: Passive positioning via WiFi APs (2.4 GHz) and BLE beacons
* LEDs

Note: In this order: BlinkUp status LED should be middle LED to simplify identification

* + Programmable status LED for user notifications (multicolor)
  + BlinkUp status LED (middle)
  + Modem netlight LED
* Operation on 3xAA or 4xAA batteries (Energizer Ultimate Lithium) for up to 5 years (based on battery type and application behavior)
* Battery level monitoring (method to be confirmed, e.g. voltage or coulomb)
* Antennas: Cellular, GNSS, WiFi/BLE (for passive reading)
* Designed for high-volume manufacturing
* Proposed housing (IP6X+ water/dust resistance) <http://www.kingwoiot.com/index.php/Product/detail/cat_en_name/nt07e.html>



**Target Hardware BOM Cost**

Estimates at 1k units: approx. $50, based on design, manufacturing, component availability, and other factors

# Device Software Requirements

Note that the listed application behavior is meant as a description of the Software Blueprint, specific additional functionality required by the customer can be added as a separate project.

## General software requirements

* Modular design and source code to support easy customization
* Initially, the lead customer portal is chosen for integration, but the integration must be modular as the open source version will likely use a different portal (e.g. a demo portal)
* Software must be free of 3rd party rights or -- if 3rd party code is used -- such code must be clearly marked and must use MIT license
* ~~Preprocessed (one-file) code must be structured so that all the configuration values are located at the top of the file. This is necessary so that when the user views the code in the IDE and wants to make configuration changes they don’t have to go hunting through the code to find the place to do that (see comment)~~
* Configuration values include
  + Ublox assist now token
  + Google Maps API token
  + Cloud REST API URL & credentials
  + Application values such as reporting frequencies, alert thresholds, etc.
* At application startup the configuration values must be inherited from user-defined (device group) environment variables. These environment variables are set in one of two ways:
  + as part of the provisioning process through the impCentral API
  + as part of manual configuration by the user (for example by editing the IDE device group settings or copying & pasting the cfg-json)

Note: Rebuild & redeployment is required for the application to pick up changed configuration values.

* At application runtime the configuration values can be modified via the southbound configuration API.
* For production software **(not PoC or Field Trial)**
  + basic developer documentation of high-level design, modules, and public APIs
  + must be ready to be published with MIT license on github
  + Note: Scope of the open source release work and documentation will be redefined/reduced for the first release due to timing/budget

## Quality Metrics

**Issue Level Definition**

For the project, the following simplified definitions of issues will be used:

* **P1: Critical.** Business critical functionality unavailable/unusable. No workaround possible.
* **P2: Degraded**. Key functionality impacted or unreliable. Workaround may be available but impacts product quality/user experience.
* **P3: General.** Other functionality impacted
* **P4: Minor.** Minor or cosmetic.

**PoC Phase**

* PoC hardware (imp006 breakout)
* PoC-level functionality: See requirements
* Basic manual testing of key functionality
* <= 2 P1 issues, <= 7 P2 issues

**Field Trial Phase**

* Initially imp006 breakout, then field trial/production hardware
* Field Trial-level functionality: See requirements
* Testing of core functionality: API automated unit testing, manual integration testing, end-to-end system and usability testing
* Zero P1 issues, <= 3 P2 issues, <= 10 P3 issues

**Production Phase**

* Production hardware
* Production-level functionality: See requirements
* Comprehensive testing: API automated unit testing, manual integration testing, end-to-end system and usability testing
* Zero P1 issues, <= 1 P2 issues, <= 5 P3 issues

## Application Functionality

|  |  |
| --- | --- |
| Phase | Description |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Device configuration state**   * Device can be remotely configured with a set of parameters * Configuration changes are queued on cloud agent. When device connects, changes are pushed to device and all applied configuration changes are acknowledged back to cloud agent (to ensure device configuration is in sync with cloud agent). * Ability to query complete state from cloud agent (to confirm current configuration) * Possible device configuration values include (TBC):   + Sensor reading period (TS)   + Normal reporting period (TR)   + Tamper detect on/off   + Movement detection (motion threshold, continuous motion duration)   + Alerts on/off & thresholds (for sensors, shock, battery, etc)   + **(not in PoC)** On-device geofence on/off & center/radius   + Tracking mode on/off & reporting period (TT)   + **(not in PoC)** Repo/theft mode on/off & reporting period/schedule   + **(not in PoC)** Known BLE beacons and locations   + **(not in PoC)** Log level |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Battery-efficient operation**   * Sleeps between operations to maximize battery life * **(not in PoC)** Shallow sleep only (to preserve black box recorder) * Cellular on only for cloud connection * Peripherals on only when needed (sensors, WiFi/BLE, etc.) |

|  |  |
| --- | --- |
| Field Trial  Production | **Robust/defensive design & operation**   * Battery safeguards   + Back-off for power-intensive operations: If a power-intensive operation (i.e. cellular connection or acquiring a GNSS fix) fails multiple times in a row then back off and retry less frequently   + Prevent crash loop from draining battery: If the application repeatedly restarts by accident (e.g. due to an error) then enter recovery mode. Recovery mode halts regular application operation and periodically checks until new application version is available. (maybe also define as “empty configuration mode” where the app does nothing except for periodically check the connection) * Connection safeguards   + Implement a minimum and maximum cloud agent connection period to ensure that misconfiguration does not result in very frequent connections or device being inaccessible for long periods. * Support Super SIM OTA updates   + Initial support for OTA updates by receiving a flag on the device that instructs the device to use forceSuperSimOTA() and then stay online for 1 minute to allow an OTA update to complete * Logging   + Device and cloud code must provide logging output of key information, warnings, and errors, enabled by log level (INFO, WARN, ERROR) to adjust verbosity   + Log level can be set by configuration parameter |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Cloud connectivity**   * Cellular only |

|  |  |
| --- | --- |
| Field Trial  Production | **WiFi/BLE support**   * ~~On PoC hardware, initially~~    + ~~1MW (with impOS support)~~ * PoC hardware and production hardware   + ESP module (with ESP-AT driver) * Need application-level ESP-AT driver, see https://docs.espressif.com/projects/esp-at/en/latest/AT\_Command\_Set |

|  |  |
| --- | --- |
| Field Trial  Production | **Shipping mode**   * Device deep-sleeps in ‘shipping mode’ after production to conserve battery * Upon removing ‘tamper detection patch’ exits ‘shipping mode’ and begins normal operation. This includes a first report with sensor values, location, battery level, etc. Note: The device never returns back to ‘shipping mode’. |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Positioning**   * Positioning is needed indoors (buildings, warehouses, or places with poor GNSS reception) as well as outdoors. * Position typically only needs to be determined when device is moving, to avoid battery drain (by repeated useless positioning of a stationary device) * Determine position in this order:   + **(not in PoC)** BLE beacons (known list of beacons + positions. Works offline. Approx. accuracy: 5m to 15m)   + GNSS (Works offline. Approx. accuracy: 3m to 20m)   + WiFi (APs & Google Location API. Approx. accuracy: 10m to 100m)   + **(not in PoC)** Cellular trilateration (tower(s) & Google Location AP. Approx. accuracy: 100m to 3,000m)   + Note: Type of positioning needs to be provided with the report * If a position is found with a method, skip the following methods for that cycle |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **GNSS**   * Use assist to minimize fix time. Load/cache valid assist data. |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Shock/Motion**   * Wake on motion above configured motion threshold * Send shock alert if acceleration above configured shock threshold * Ongoing motion means position can/should be reacquired. Ignore sporadic/brief motion, i.e. no position change * Tracker should not continuously be awake during motion (to preserve battery) |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Offline operation**   * If no network connection, data & events are stored for a max of X days on device and reported to cloud agent when connection is regained |

|  |  |
| --- | --- |
| Field Trial  Production | **On-device geofencing mode**   * Goal of on-device geofence is to preserve battery by avoiding unnecessary position reporting while inside the geofence * Enabled via configuration setting from cloud agent * One geofence (center + radius) is sent from cloud agent to device to activate geofence mode * Geofencing must support offline positioning modes, i.e. GNSS positioning as well as BLE beacon positioning **(not in PoC)**. (Note: This may mean the geofence must be implemented in the application rather than leveraging GNSS built-in geofence). * If geofence mode is enabled, a position report is sent if the device enters or exits the geofence. Note: Tracking mode is not automatically enabled … see also tracking mode. * Note: Additional (off-device) geofencing can still be done on tracking portal based on tracking reports |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Periodic sensor reading**   * Periodic (every TS seconds, configurable) wake * Read sensor values (temperature, battery level, etc) * Check for alerts/thresholds. If alert is required, report to cloud agent (see alerts) * If no alert is required, queue report for next connection |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Alerts**   * Values are checked on every wake * If alert condition is detected (sensor value is exceeded, tampering (light sensor is triggered), send alert to cloud agent immediately (semi-real time) * Send further alerts to cloud agent only if conditions have changed (i.e. do not automatically repeat previous alerts) |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Normal reporting mode**   * Periodic (every TR seconds, configurable) report of queued information to cloud agent * Includes current position (use last position if no movement occurred) |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Tracking mode**   * Enabled via configuration setting from cloud agent. Upon enabling, acquire position and send a position report to report the current position (even if the device is not moving) * On movement, periodically (every TT seconds) acquire position and send position report * When movement stops, send another position report to report the current position * If on-device geofence is enabled, do not send position report while inside geofence |

|  |  |
| --- | --- |
| Field Trial  Production | **Repo (“Repossession”) / Theft mode**   * Repo / theft mode is like tracking mode, but activated by a previously defined schedule, e.g. automatically activate tracking mode on May 1st. * (Use case: Asset is expected to be returned on April 30, so starting May 1st it is automatically tracked) |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Northbound API & application data** (from cloud agent to tracking portal)   * Periodic reporting: Temperature, battery level, position, etc * Alerts: Tamper, temperature, battery level, position, etc * Current device configuration state * APIs & JSON schema: See section “Cloud Agent Software” below |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Southbound API & application data** (from tracking portal to cloud agent)   * Configuration values * APIs & JSON schema: See section “Cloud Agent Software” below |

|  |  |
| --- | --- |
| PoC  Field Trial  Production | **Device setup/commissioning**   * **(PoC only)** Electric Imp blinkup app to associate user * **(Field Trial/Production)** tracker ID/QR code + app to associate user |

|  |  |
| --- | --- |
| Production | **Device Evaluation UI**  Device agent provides simple web UI for checking device status and allowing configuration (to avoid user having to check logs, hit the APIs, or modify code)   * **Status page:** Display subset of data contained in the last report (trackerId, status, location, sensors, alerts) * **Configuration page:** Ability to change configuration subset (GOOGLE\_MAPS\_API\_KEY, UBLOX\_ASSIST\_NOW\_TOKEN, DEFAULT\_DATA\_READING\_PERIOD, DEFAULT\_DATA\_SENDING\_PERIOD, DEFAULT\_TEMPERATURE\_HIGH, DEFAULT\_TEMPERATURE\_LOW, DEFAULT\_SHOCK\_THRESHOLD, DEFAULT\_LOCATION\_READING\_PERIOD, MOVEMENT DETECTION PATTERN: DESK, WALKING) * Ability to disable UI in production deployments (where it is not needed)   Note: MOVEMENT DETECTION PATTERN is a simplified flag that updates the multiple internal configuration values for the respective pattern |

|  |  |
| --- | --- |
| Field Trial  Production | **ESP32 firmware update**   * Ability to deliver and verify DFU firmware update to local ESP32 module * Firmware image is provided in one of two ways   + For manufacturing: From preloaded local flash file system   + For in-field updates: From cloud REST API (via agent)   Note: Our understanding is that ESP32 chips comes blank, so firmware update is high priority for the Field Trial (pre-production) hardware |

|  |  |
| --- | --- |
| Field Trial  Production | (Feature request, to be discussed for implementation)  **Wireless information to be included in reports and logs**   * Cellular signal quality (i.e. RSSI, RSRO/RSRQ as available) * Number of visible GNSS satellites |

## Device Evaluation UI (to be finalized)

Additional requirements/information about the above mentioned Device Evaluation UI, to capture the status of the latest discussions:

* The UI will use the UX platform and design principles as much as possible/practical given the limitations of the asset tracker product, platform properties, screen sizes, etc.
* Ease of design and implementation is prioritized over elegant design.
* The UI must support recent mid- to large-size phone screens as well as desktop screens, but layout and resizing does not need to be perfect on either.
* For the location (map) display a free map service can be used, this avoids requiring the user to acquire and enter a Google Maps key
* Alerts do not need to be displayed or represented by icons on the status page (because that would require stateful alert logic in the agent or UI). It is sufficient to refer to the log for alerts.
* The log screen should contain the last 10 events.

## Library functionality

Squirrel libraries to be created/updated as part of the work:

* New: ESP-AT driver
* New: Library for tracking portal integration, including simple REST API simulator (exact functionality/portal to be defined)
* ~~New: Library to manage Super SIM OTA updates~~
* ~~Update: BG96 GNSS library~~
* Update: ublox GNSS library, using ublox assist (as needed)
* Update: Google Maps library (cellular positioning)
* Update: ReplayMessenger library (as needed)
* ~~Update: Low-power manager library (to support shallow sleep)~~

Note: Same quality requirements as the rest of the code.

## Out of Scope

* Companion mobile app

We’re trying to avoid a dedicated mobile (BlinkUp) app since all devices have cellular and can be pre-provisioned in the factory. So the only thing left in the field (by the customer/user) is to associate a tracker with a user. This can be done via tracker ID on label or QR code + web app (hopefully a dedicated mobile app is not needed).

# Cloud Agent Software

Initially, the lead customer portal is chosen for integration, but the integration must be modular as the open source version will likely use a different portal (e.g. a demo portal). I.e.:

* Integration with lead customer tracking portal: APIs & schema see below “Simplified Proposal for Cloud Integration”. Note: If AWS integration is required it is expected that existing Electric Imp AWS libraries are leveraged.
* Note: Customer currently has a node.js application running to handle incoming device data and forward to DynamoDB/Postgres. Customer is in the process of reworking the architecture and redefining the APIs and data model, so adjustments are possible. Customer to provide example APIs.
* For open source version: TBD. Possible candidates: <https://www.traccar.org> or https://ubidots.com/

## Simplified Proposal for Cloud Integration

Simplified proposal for initial integration, either into existing v1 cloud or upcoming v2 cloud.

**Data sending to the cloud (Northbound API & JSON data model)**

REST API on the cloud. To be implemented by the customer.

* One endpoint should be enough for the beginning (exact endpoint name can be changed): https://<cloud\_api\_url>/data
* POST request from imp to send new portion of data.
* Basic authentication (username/password). Can be substituted by an API Key later.
* Body with data in text (JSON) format.
* Response code 200 from the cloud if the request/data is accepted.
* Some fields (keys) in the JSON can be mandatory, some - optional (do not exist in every message) - depends on the current configuration and/or on the event which triggered the message sending.

An example of data msg (all fields are just for example. The exact format/content/names TBD during design/implementation):

{

"trackerId": "c0010c2a69f088a4",

"timestamp": 1617900077,

"status": {

"movement": true,

"tracking": true

},

"location": {

"type": "gnss",

"accuracy": 3,

"lng": 30.571465,

"lat": 59.749069

},

"sensors": {

"batteryLevel": 64.32,

"temperature": 42.191177

},

"alerts": {

"temperatureHigh": true

}

}

**Configuration (Southbound API & JSON Data Model)**

REST API on imp-agent. To be implemented.

* URL of every imp-agent is known after the imp enrollment.
* A couple of requests/endpoints for beginning (exact number and names can be changed):
  + PATCH/POST https://<imp\_agent\_url>/cfg - pushes a configuration update from the cloud to the imp.
  + GET https://<imp\_agent\_url>/cfg - returns the latest known actual configuration from the imp to the cloud.
* Response code 202 (TBD) from the imp-agent if the request is accepted (does not mean it is executed).
* If/when imp-device is online, cfg update requests are forwarded from imp-agent to imp-device for execution.
* The cloud can periodically poll the imp-agent (GET request) to obtain the actual known configuration.
* Basic authentication (username/password).
* Body with data in text (JSON) format.
* Some fields (keys) in the JSON can be mandatory, some - optional.
* Configuration update requests can contain full configuration or only parts/fields of configuration which should be updated (assuming these are logically consistent parts).

An example of cfg update request (all fields are just for example. The exact format/content/names TBD during design/implementation):

{

"cfgFormatVersion": "1.0",

"readingPeriod": 600,

"reportingPeriod": 43200,

"temperatureHigh": 40,

"tracking": false

}

* The actual cfg returned to the cloud is always the full configuration.

# Device Manufacturing & Test Software Requirements

The manufacturing and test software leverages the Electric Imp connected manufacturing process.

|  |  |
| --- | --- |
| Field Trial  Production | **Manufacturing & Test software to be developed**   * Device hardware testing code & PASS/FAIL recording * Printing of individual label with device information:   + IMEI   + ICCIDs   + Tracker ID serial number (imp deviceID)   + Tracker ID QR code * Device blessing (hardware activation) -> ready to ship |

# Appendix: Planned Testing Checking by Customer

* Device successfully registers on cell network
* Device successfully communicates with our server
* ~~Device accepts and responds to all SMS commands~~ (no SMS support)
* Device accepts and responds to all ~~TCP~~ commands
* Device accepts configuration modifications
* Device retains configuration modifications after a soft reboot
* Device retains configuration modifications after a hard reboot
* Device reports consistently at the defined interval
* Device backfills all buffered data when a broken connection is resumed
* Device isn't prone to random reboots or crashing firmware
* Device is able to report a GPS location when installed on a trailer
* Device is able to report a GPS location when installed underneath a vehicle
* Device is able to maintain a cellular connection consistently during a shipment
* Device is able to maintain a consistent cellular connection when indoors
* Device is able to detect nearby wifi access points and successfully reports MAC addresses to server
* Device consistently sends alerts ~~via SMS and/or TCP~~ for each configured event
* Device wakes after removing ‘tamper detection patch’
* LED indicator functions properly
* Device reports at least as accurately and consistently as the Micron Bolt Mini when devices are installed on the same asset
* Battery level can be accurately estimated
* Battery life isn't affected by poor connectivity

# Addendum: Field Trial 2 Change Requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Area | Priority | Description | Response (NBL) | Effort Estimate |
| Agent Web UI | Med | (Ishita) Feedback: <https://www.figma.com/proto/mXdcAHsnda3ezfnfCk5jEB/Programmable-X---Asset-Tracker?page-id=1198%3A121544&node-id=1386%3A123936&viewport=232%2C148%2C0.13&scaling=scale-down&starting-point-node-id=1386%3A123936&show-proto-sidebar=1> |  |  |
| Agent Web UI | Med | (Terrence) Automatic or manual refresh of dashboard page (to see changes sent by device) |  |  |
| Agent Web UI | Low | (Terrence) Visible feedback that device has applied configuration update (beyond watching updateId) … Edit: It sounds like the “**Last updated on:**” section in the UI (top right) is sufficient for telling the user when the update was applied. |  |  |
| Default configuration via env variables | High | (Terrence) Inherit default config values from env variables rather than hard coded in the sources (see comment above *“@alexey.kosenchuk@nobitlost.com Updated the sections related to configuration, using env variables. Please review.”*) |  |  |
| Agent Web UI HTML/JS Open Source | High | (Terrence) UI of agent must also be made available as open source, no extra documentation required. |  |  |
| traccar integration | Med | (Terrence) Simple integration into traccar. See below. |  |  |

**traccar integration:**

traccar.org is not a great tracking app but it’s readily available to try for free via one of their demo instances (<https://www.traccar.org/demo-server/>). It is desirable to provide a simple out-of-the-box integration.

The easiest integration appears to be via the OsmAnd protocol adapter:

<https://www.traccar.org/osmand/>

for implementation details, see [https://github.com/traccar/traccar/blob/master/src/main/java/org/traccar/  
protocol/OsmAndProtocolDecoder.java](https://github.com/traccar/traccar/blob/master/src/main/java/org/traccar/protocol/OsmAndProtocolDecoder.java)

I tried this by modifying the agent CloudClient.send() function:

local query = "?deviceid=" + body.trackerId +

"&lat=" + body.location.lat +

"&lon=" + body.location.lng +

"&accuracy=" + body.location.accuracy +

"&batt=" + body.sensors.batteryLevel;

local req = http.post("http://demo3.traccar.org:5055/" +

query, {}, "");

This worked. Unfortunately, the OsmAnd integration only appears to support a very limited set of telemetry values such as deviceId, lat/lng, location accuracy, and battery level.

Integration proposal:

* In agent web UI, add “traccar” into Cloud Rest API drop-down
* User can insert the Cloud REST API URL for the traccar demo server (username and password are ignored)
* When “traccar” is selected, agent uses code similar to above CloudClient.send() code to send telemetry to traccar
* Note: Integration of configuration settings not required/possible (configuration still needs to be done via agent web UI or env variables)