GPS coordinates exposed by EVE

Motivation

The requirement to obtain device location coordinates from a GPS (or more generally GNSS) receiver and publish them to the controller (and possibly also to applications) comes from an EVE customer. They need to be able to track locations of their outdoor deployments (with much higher precision than what IP-based geolocation can provide). This should be possible without the need for any additional hardware because the target device configuration includes a *Sierra Wireless EM7565* LTE modem with an integrated GPS receiver. However, it is preferred that the method to obtain location coordinates is not specific to this hardware and instead can support a wide range of GPS-enhanced LTE modems. This can be possible by use of (de facto) standardized protocols, such as QMI/MBIM to control the modem and NMEA for formatting of GNSS data.

External (or built-in but not integrated with LTE) GNSS receivers will not be covered by this proposal.

LTE Modem Interface

Almost all well-known cellular modems get hooked to a USB bus. From there, a single device will expose different interfaces to accommodate various ways of interacting with it (each appearing as a separate `/dev/ttyUSB*` device). There is always a traditional serial interface accepting AT commands. A GPS-capable modem will expose AT commands such as `GPSAUTOSTART, GPSATINFO` and `GPSLOC` to start location tracking and to obtain location data. The problem with AT commands is that they are not standardized and differ from modem to modem, therefore EVE avoids using the AT interface.

Additionally to AT commands, modern cellular modems expose CDC-based MBIM and QMI interfaces with well-defined APIs. EVE OS (specifically wwan microservice) uses CLI clients provided for these protocols - `mbimcli` and `qmicli`, to control the modem connectivity status, get state data as well as metrics. Even though the aim is to support both interfaces and MBIM is more of a standard, EVE prefers QMI simply because it uses a more reliable NCM protocol underneath. Also, `qmicli` is much more feature-rich than `mbimcli`, including in the area of location tracking as described below. Note that *Sierra Wireless EM7565* supports both QMI and MBIM and allows switching between the two.

Lastly, GPS-enhanced cellular modems will provide additional serial interface streaming NMEA traces with location data (coordinates, satellite information, etc.). This is typically used in combination with AT commands in older applications. However, QMI/MBIM protocols also cover publishing of location data and serial ports are not needed anymore.

Location Service

Focusing now on QMI, the protocol uses request-response communication pattern between a modem and a user-space client for the modem management, and additionally allows to stream so-called *indications* from the modem with state updates, such as NMEA traces with location information.

The QMI protocol defines several services, each of them related to different actions that may be requested. For example, WDS (Wireless Data) service allows setting up IP connections. The service of interest for us here is *LOC*, aka *location service*, allowing to start location tracking and retrieving or subscribing for location status information, which are obtained either as raw NMEA traces or as parsed location coordinates (latitude, longitude, etc.). The location tracking is started using a request-response communication, while location updates are streamed as indications. A full set of messages of the LOC service can be found in the libqmi repository.

Here we can see the set of attributes available inside a position update indication. Note that for this feature request the latitude and longitude are required, while altitude and other attributes are optional (*as confirmed by the customer*, but see below what other attributes we suggest to also include).

`qmicli` tool makes interaction with the LOC service very easy. Location tracking can be started using `--loc-start` command option, while position updates can be watched for with `--loc-follow-position-report`. A non-blocking alternative that allows to get a single report for the current location can be obtained with `--loc-get-position-report`.

For example (note the use of client ID to match request with response):

```
# qmicli -p -d /dev/cdc-wdm0 --loc-start --client-no-release-cid
[/dev/cdc-wdm0] Successfully started location tracking (session id 0)
[/dev/cdc-wdm0] Client ID not released:
    Service: 'loc'
        CID: '1'
# qmicli -p -d /dev/cdc-wdm0 --loc-get-position-report --client-cid=1
[position report] status: success
   latitude:  37.334464 degrees
   longitude: -121.894300 degrees
   circular horizontal position uncertainty:           25.000000 meters
   horizontal elliptical uncertainty (semi-minor axis): 12.000000 meters
   horizontal elliptical uncertainty (semi-major axis): 21.000000 meters
   horizontal elliptical uncertainty azimuth:          84.375000 meters
   horizontal confidence: 39%
   horizontal reliability: medium
   horizontal speed: 0.000000 m/s
   speed uncertainty: 0.657951 m/s
   altitude w.r.t. ellipsoid: 79.643311 meters
   altitude w.r.t. mean sea level: 107.592033 meters
   vertical uncertainty: 16.000000 meters
   vertical confidence: n/a
   vertical reliability: medium
   vertical speed: 0.000000 m/s
   heading: n/a
   heading uncertainty: n/a
   magnetic deviation: 15.000000 degrees
   technology: satellite
   position DOP:   1.300000
   horizontal DOP: 1.000000
   vertical DOP:   0.800000
   UTC timestamp: 1647428316000 ms
   Leap seconds: 18
   GPS time: 2201 weeks and 298734000ms
   time uncertainty: n/a
   time source: navigation-solution
   sensor data usage: (null)
   Fix count: 1
   Satellites used: 1,4,21,32,67,78,83,195,211,212,308,315
   Altitude assumed: yes
```

What's more, `qmicli` also imports libmbim and to some extent is able to talk to modems in the MBIM mode. This does include the location service and allows us to use the same CLI command regardless of the underlying protocol (the use of the MBIM protocol over QMI can be enforced with `--device-open-mbim`).

This is important not only because it simplifies the implementation, but also because `mbimcli` is somewhat behind in development and does not yet provide commands for the interaction with the location service.

EVE API Additions

First of all, it should be possible to turn the location tracking ON and OFF from the controller and by default it should be disabled. We will add a new `location_tracking` boolean flag into `CellularConfig` (a place for LTE modem config, such as APN):

```
message CellularConfig {
  // APN string - by default it is "internet"
  string APN = 1;
  // Optional cellular connectivity probing.
  // By default it is enabled.
  CellularConnectivityProbe probe = 2;
  // Some LTE modems have GNSS receiver integrated and can be used for
  // device location tracking.
  // Enable this option to have location info periodically obtained from
  // this modem and published to the controller and to applications.
  bool location_tracking = 3;
}
```

EVE API already defines GeoLoc structure for device location information:

```
// From an IP address-based geolocation service
// XXX later define GPS coordinates from device
message GeoLoc {
  string UnderlayIP = 1;
  string Hostname = 2;
  string City = 3;
  string Region = 4;
  string Country = 5;
  string Loc = 6;
  string Org = 7;
  string Postal = 8;
}
```

This structure was originally defined to carry device location data determined based on a public IP address of the device (using online service http://ipinfo.io). As such, there is an instance of `GeoLoc` for every device network inside `ZInfoNetwork`.

With IP-based geolocation, the estimated latitude and longitude are practically useless and typically point to some random place in the determined city. For this reason, not much attention was given to the `Loc` field and its type. It was defined as a string and requires parsing when used for more than just display (content example: Loc:"48.7395,19.1535").

For a much more precise GNSS-based location tracking we propose to define latitude, longitude (and altitude) as separate fields with double-precision floating-point types, encoding coordinates with decimal degrees (DD) notation. An open question is whether the address information, such as City, Region, Country, etc., should be provided in this case as well. It would require a use of an online Reverse geocoding service to obtain this data. If not required (which we will assume unless told otherwise), then it does not make much sense to reuse the GeoLoc structure at all, since all existing fields would be left empty in this case.

Answer from customer: Reverse geocoding is not required, although it would be a nice-to-have feature. But for now this will be out-of-scope of this proposal.

We propose to define a separate structure for GNSS location information:

```
enum LocReliability {
  LOC_RELIABILITY_UNSPECIFIED = 0;
  LOC_RELIABILITY_VERY_LOW = 1;
  LOC_RELIABILITY_LOW = 2;
  LOC_RELIABILITY_MEDIUM = 3;
  LOC_RELIABILITY_HIGH = 4;
}

// Geographic location of the device obtained from a GNSS receiver.
message ZInfoLocation {
  // Latitude in the Decimal degrees (DD) notation.
  double latitude = 1;
  // Longitude in the Decimal degrees (DD) notation.
  double longitude = 2;
  // Altitude w.r.t. mean sea level.
  double altitude = 3;
  // UTC timestamp for the location measurement
  // (recorded by the GNSS clock).
  google.protobuf.Timestamp utc_timestamp = 4;
  // Reliability of the provided information for latitude and longitude.
  LocReliability horizontal_reliability = 5;
  // Reliability of the provided information for altitude.
  LocReliability vertical_reliability = 6;
  // Circular horizontal position uncertainty in meters.
  float horizontal_uncertainty = 7;
  // Vertical position uncertainty in meters.
  float vertical_uncertainty = 8;
}
```

`ZinfoLocation` will be added into the list of info messages. Unlike other info messages, which are published on every change, here we may want to set a fixed (and configurable) publish period (config item: `timer.location.interval`). Note that we will get location updates from the receiver one every second (at most), but our default period will be longer and in the range of tens of seconds (e.g. publish every 20 seconds), so that the controller will not get overloaded with too many updates (from all devices).

Additionally, we could expose the location information to applications. An open question is whether we need to provide this information to any application running on the device, or if it is sufficient to only post it to the (single) application designated as Local Profile Server.

Answer from customer: Every application should be able to retrieve location information.

We will add a new POST endpoint `/api/v1/location` for the Local profile server, where EVE would periodically `POST` the content of the `ZInfoLocatio` proto message. Default period for these APIs is 1 minute. For location information we would apply the same period as configured for publishing to the controller.

For other applications, we will leverage the meta-data server created for every network instance (under IP 169.254.169.254) and add a new GET endpoint `/eve/v1/location.json`, providing json-encoded `ZInfoLocation` on demand.

EVE implementation changes

To implement this feature, we need to extend the wwan microservice with a function/process that will periodically collect and publish location information. This will be done using `qmicli` as outlined above. With further testing and experimentation we will determine if it is more reliable to watch for location indication messages (with a process running in the background), or if the microservice would rather periodically ask for a location update (with a non-blocking call, just like it is done for connection status and metrics).

Conclusion: It is more efficient and reliable to continuously watch for location indication messages using a background process.

The wwan microservice exposes state information and metrics as `/run/wwan/status.json` and `/run/wwan/metrics.json`, respectively, which are then picked up by the NIM microservice and published further using pubsub.

It might make sense to publish location information as a separate json file, for example as `/run/wwan/location.json`. Similarly like metrics and status, location updates will be forwarded further (at a limited rate) by NIM to other microservices via pubsub.

Zedagent will subscribe to location reports and publish them periodically to the controller and the Local Profile Server. Zedrouter will also collect location information from wwan microservice and make it available to all applications on demand.