Telit K3 AGNSS EE Injection Application Note

80434NT11835A Rev. 0 – 2020-09-16

TELIT
TECHNICAL
DOCUMENTATION

APPLICABILITY TABLE

PRODUCTS

- SL871
- SL871-L
- SL869-V2
- SL869L-V2
- SE868-A
- SE868K3-A
- SE868K3-AL
- SE878K3-A
- SC872-A
- SC874-A

CONTENTS

I

TABLE LIST

FIGURE LIST

1. INTRODUCTION

1.1. Scope

This document describes the Extended Prediction Orbit (EPO) GNSS feature supported by the Telit's MT33xx flash-based GNSS modules (see Applicability table above for the complete list) and provides details on the EPO update process.

The EPO-II file format and the EPO update process protocol, designed and defined by Mediatek®, are hereby described in support of typical user applications.

It is useful for those who are interested in designing and implementing a host system that communicates with the Telit's GNSS module families and provides the EPO update functionality.

1.2. Audience

This document is intended for public distribution to potential customers who are evaluating a GNSS module from the V13 firmware family listed in the Applicability Table. It can also be used by customers who are interested in designing and implementing a host system that communicates with the Telit's GNSS module families and want to integrate the EPO update functionality.

1.3. Contact Information, Support

For general contact, technical support services, technical questions and report documentation errors contact Telit Technical Support at:

- TS-EMEA@telit.com
- TS-AMERICAS@telit.com
- TS-APAC@telit.com

Alternatively, use:

https://www.telit.com/contact-us

For detailed information about where you can buy the Telit modules or for recommendations on accessories and components visit:

http://www.telit.com

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

Telit appreciates feedback from the users of our information.

1.4. Text Conventions

	Danger – This information MUST be followed or catastrophic equipment failure or bodily injury may occur.
	Caution or Warning – Alerts the user to important points about integrating the module, if these points are not followed, the module and end user equipment may fail or malfunction.
	Tip or Information – Provides advice and suggestions that may be useful when integrating the module.

All dates are in ISO 8601 format, i.e. YYYY-MM-DD.

1.5. Related Documents

[1] Telit SE868xx-A Family Product User Guide, 1VV0301201
[2] Telit SL871 Family Product User Guide, 1VV0301170
[3] Telit SL869x-V2 Family Product User Guide, 1VV0301175
[4] Telit SC872-A Product User Guide, 1VV0301202
[5] V13 Software User Guide, 1VV0301162
[6] V13 Software Authorized User Guide, 1VV0301550

2. EXTENDED EPHEMERIS

2.1. Overview

Extended Ephemeris (EE) refers to a technology developed by GNSS chip vendors to reduce Time To First Fix (TTFF), particularly in challenging RF environments affecting satellite signal reception.

Telit and GNSS chip vendors collaborate to provide GNSS technology solutions that can be used on Telit GNSS modules to reduce the Time-To-First-Fix (TTFF).

2.2. Background

For a GNSS receiver to provide a navigation solution, it must have satellite orbits data so that it can accurately determine the expected location of each visible GPS satellite. If the receiver does not have this data, or if the data it has is too old, it must collect the data from the satellite after its signal is acquired. Even under the best circumstances, where initial position and time are known, this process can take up to 35 seconds, which in turn can mean a Time To First Fix (TTFF) of 35 seconds or more. In typical circumstances the TTFF can be even longer, as the receiver must collect ephemeris for three or four satellites in order to navigate.

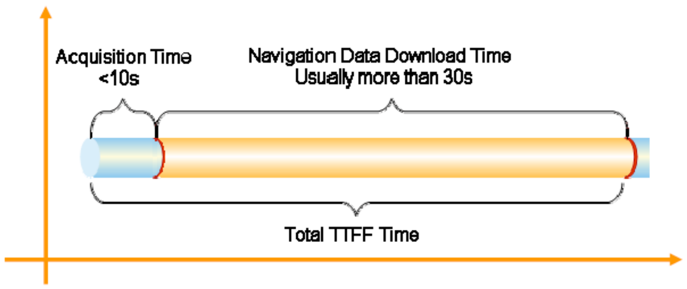


Figure 2-1 TTFF Time Breakdown

EPO technology provides pre-calculated synthetic Extended Ephemeris (EE) data to the GNSS receiver to reduce TTFF. The presence of Extended Ephemeris allows the receiver to substitute it for broadcast ephemeris downloaded from the GNSS satellite and to skip the navigation data collection requirement.

2.3. Telit EPO Technology

Telit EPO technology includes the calculation of synthetic EE, delivery and use by a supporting GNSS module in order to help a receiver to calculate position solution and reduce TTFF performance.

EPO are calculated, at a remote source (that is at a vendor’s AGPS server), by applying predictive models of satellite motion to actual broadcast ephemeris data and is structured as a set of data segments on combination of satellite constellations; they can be GPS only, or GPS + GLONASS.

For Telit EPO EE, utilized by the GNSS modules referred by this document (see applicability table), each segment contains synthetic ephemeris for a 6-hour time period. Thus, the extended ephemeris data for one day consists of 4 segments.

2.3.1. EPO Calculation by Servers

The Telit EPO server obtains ephemeris information from world-wide GNSS tracking stations and calculates EE data for all healthy GNSS satellites and store the data at on-demand basis.

The EE-EPO data has following characteristics:

- The EE data calculation and update are done at a predetermined schedule that is adequate to provide valid data at any time.
- The server calculates a sufficient number of data blocks to produce EE and the data is stored in files which can be accessed over a network connection.
- The EE data blocks are packaged into files according to on different prediction intervals. Each of the files represents a different prediction interval such as one-day, three-day, seven day, 14 days, and so on which is the applicable length of time (in days) over which the EE within the file can be used by the receiver.

2.3.2. EPO Delivery to Receivers

To use EE-EPO data, an OEM device (either a hosting device or functions like a host), must be capable of establishing an HTTP connection and retrieving data files from an EE-EPO server.

The OEM device must also transfer the file to the OEM GNSS receiver module (operating as a client) over the communication link between the host and the GNSS device (that is, a serial port).

2.3.3. EPO Usage Setup

The use of EE EPO data by the receiver module requires that precise time is first derived and verified from live satellites. Thus, the use of EE to reduce TTFF is most effective for applications in which the time on a GPS receiver module is maintained in the Hibernate state.

Once the module has navigated, time is verified and updated in the Real-Time Clock (RTC), where it is maintained on the module. Thus, at start-up the module leaves its initial state with an accurate initial time, as well as an initial estimate of position from memory. The initial time eliminates the need to derive time from a live satellite in order to navigate, and if broadcast ephemeris has expired or is invalid (that is Warm Start conditions), the module uses EE instead. Under Warm Start conditions the typical TTFF will be approximately 10 seconds or less in good RF environments (minimal blockage and higher signal-to-noise ratios).

However, if the receiver module starts up from a powered off state, that is Cold Start conditions, the receiver module does not have an estimate of time. Therefore, it must derive and verify date and time and download ephemeris from the satellites, which can take up to 35 seconds, even in good conditions. Thus, the use of EE EPO data is generally not effective in reducing Cold Start TTFF.

2.4. Telit EPO Infrastructure

Figure 2-2, illustrates the overall data flow for Telit EE-EPO.

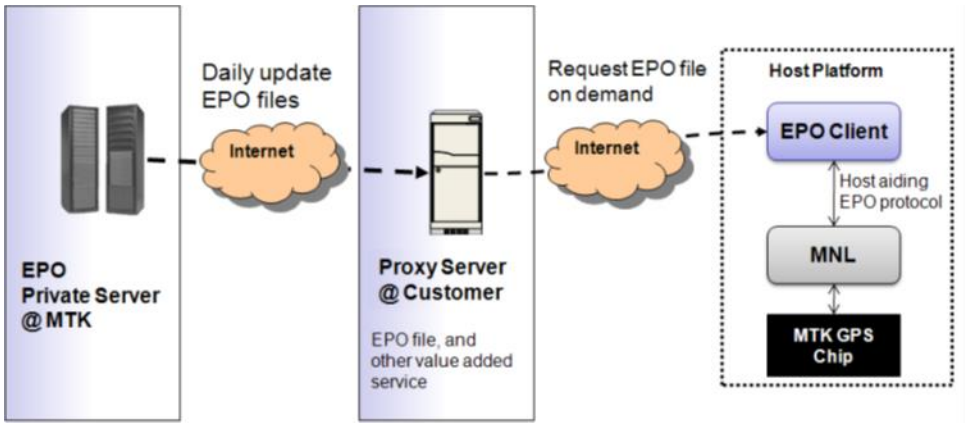


Figure 2-2 Telit EE EPO Data Flow

Refer to the above illustration for the overview of Telit EE EPO workflow, the subsections below provides more details.

2.4.1. MediaTek EPO Server

The MediaTek's Extended Ephemeris (EE) data is generated at an EPO private server using one of their proprietary predictive models with respect to the predictive interval (thar is a 30-day EE). The time frame for each new EE data is each UTC day.

For a 30-day EE interval, a number of EE segments covering the next 30 days are generated. The EE data is compressed, formatted and published as a set of EE files. All the satellites most recent ephemerides are used to calculate the EE data. Thus, all the EE within the files is new (less than one hour old) when they are published at the beginning of the UTC day.

2.4.2. Extended Ephemeris Files

Currently the EE EPO data file that is generated at the Mediatek server, publishes an EPO file for a 30-day prediction interval. However, MT3333 does not support 30-day EPO files and has a maximum storage capacity for 14 days.

For GPS constellation (32 GPS satellites), the EPO file size is 2304 bytes per segment; each segment covers a six-hour period of a day. The following table illustrates the EE EPO data validity period and the EE EPO file size. AAE EPO data filed.

EPO VALIDITY PERIOD	EPO DATA FILE
ONE SIX-HOUR	2304 (2KB+)
ONE DAY	9216 (9KB)
THREE DAY	27648 (27KB)
SEVEN DAY	64512 (63KB)
14 DAY	129024 (126KB)
30 DAY	276480 (270KB)

Table 2-1 EPO File Validity versus Size

This information provides the essential data on some fundamental considerations of use of EE EPO in positioning applications, such as the frequency of download from the EE EPO server to GNSS module, the data file size of the download, and the memory requirement (storage type and the size) in the custom device.

2.4.3. Telit EPO Server

The Telit EPO server retrieves EPO data files from the MediaTek's EPO server several times a day to ensure that the most recently updated files are available to Telit customers. Although, Telit provides no specifications with regard to availability and quality of service, it maintains redundant servers for added reliability of service.

The server URL is http://epo.telit.com. A username/password is required for access, which is provided by Telit when the customer has signed a Telit Service Agreement.

2.4.4. Customer Mirror Server

The customer mirror server retrieves EPO data files from the Telit EPO server and makes them available for the customer's deployed devices. Telit provides an EE Mirror Server Setup Application Note containing instructions for setting up this server. The customer may set up one or more additional mirror servers for purposes of redundancy and/or load balance.

2.4.5. Customer Device

The Host application within the customer device is responsible for downloading the EPO data file from the customer mirror server and transfers the file to the GNSS receiver module using the communication link between the host and the GNSS device (that is a serial port).

The GNSS device receives the file over the communication link and unpacks the EE data. The module saves the blocks of EE and stores in NVM, or saved in memory, from where they can be retrieved as needed by the GNSS receiver.

3. EPO DATA FILE TRANSFER

3.1. EPO-II Format

The basic unit of an EPO file is SAT Data, which is corresponding to the predicted satellite orbits data for one satellite in the GPS constellation, or any other constellation in the GNSS system.

As a reference, the following information is for GPS only EPO.

- Data size of a SAT Data is 72 bytes.
- One EPO segment contains the SAT Data for whole satellites in a constellation (GPS: 32), the data size for an EPO segment is 2304 bytes.
- Each EPO file contains several EPO segments; therefore, the file size must be a multiple of 2304. That means no partial segment is allowed in an EPO file.
- An EPO segment is valid for 6 hours. Therefore, there will be four (4) EPO segments contained in an EPO file to provide a predictive satellite orbits data for one day period.

For more information about the covering periods and file sizes, please refer to section 2.4.2 Extended Ephemeris Files.

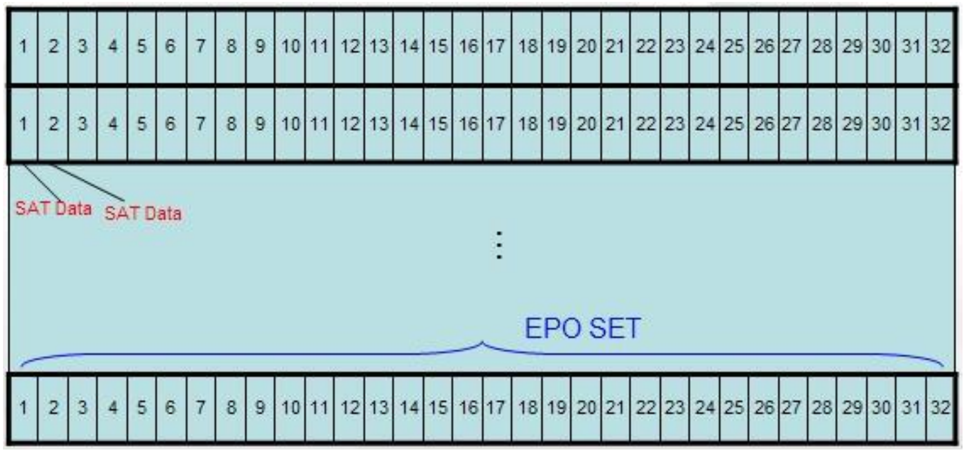
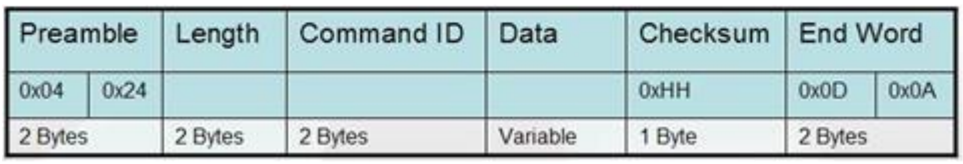


Figure 3-1 EPO File Format and Segments

An EPO SET is composed by the EPO segments for the available constellations.

3.2. Binary Protocol

This section provides a brief introduction to the MediaTek's binary packet format as it relates to the data format used in EPO file transfer.



Preamble		Length	Command ID	Data	Checksum	End Word	
0x04	0x24				0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	Variable	1 Byte	2 Bytes	

Figure 3-2 Binary Format

As it can be seen in Figure 3-2 Binary Format, each packet in MediaTek's binary packet format has the following fields:

1. Preamble (2-Byte word): 0x2404
2. Length (2-Byte word): Total number bytes in the packet from Preamble to End Word.
3. Command ID (2-Byte word):
   a. 0 ~ 999: Conform to ASCII Protocol
   b. 1000 ~ 65535: Designated for Mediatek Binary Protocol

4. Data field (Variable field): Additional data or parameters (if required) for the specified Command ID.
5. Checksum (1-Byte word): The checksum is the 8-bit exclusive OR of all bytes in the packet between but not including the “Preamble” and the “Checksum”.
6. End Word (2-Byte word): 0x0A0D

Each MediaTek’s binary packet has a maximum packet size of 256 bytes, must use little endian and use one-byte alignment.

3.2.1. EPO Data Binary Packet

The Mediatek binary packet that is used for EPO data transfer is called MTK_BIN_EPO packet (packet type 723, 0x02D3).

Field	Example	Description
Preamble	0x0424	Mediatek binary packet preamble
Length	0x00E3	The length of the payload of the binary packet
Command ID	0x02D3	Mediatek MTK_BIN_EPO packet identifier
Data	-	The Data field will contain a 2-byte EPO SEQ and three 72-byte SAT Data sub fields: - The EPO SEQ sub field is used for synchronization among MTK_BIN_EPO packets and will be an integer starting from 0x0000. - The SAT Data sub fields are filled with data gathered from input EPO data file. If there’s no enough EPO data to fulfill all the three SAT Data fields, the remaining fields can be padded with 0x00. After all EE EPO data is transferred, Host application sends a final MTK_BIN_EPO packet with sequence number of 0xFFFF to indicate the completion of the transfer.
Checksum	0xHH	An example of checksum calculated from the payload
End Word	0x0D0A	Mediatek binary packet end word

Table 3-1 MediaTek Binary Packet Format

3.3. EPO File Transfer

This section provides further details regarding the transfer of an EE EPO data file from the Host application to the receiver module.

3.3.1. EPO Transfer Requirements

As mentioned above, EE EPO data files can be stored in non-volatile memory (NVM) if it is supported. It is required that the NVM storage and/or adequate memory resource is provided for EPO technology to be operational. Currently, MT3333 supports 14-day EE EPO data files.

Even though EPO technology is also used by the receiver module when it is operating in a low power mode, it is required that the receiver is at Full-Power mode for an EE EPO data file transfer. If the receiver is operating in a low power mode, the Host application must command the module to enter full power mode prior to initiating an EE EPO data file transfer.

3.3.2. EPO Transfer Process

The process for transferring an EE EPO data file to the receiver module is performed using serial data messages to transport the file over the serial host port.

During normal operating state, the communication protocol of the GNSS receiver is set in ASCII mode in that the data input/output is on a NMEA string basis whereas in Mediatek Binary mode, that is used for EE EPO data file transfer, the data input/output is based on Mediatek Binary Protocol packets.

To start the procedure, the Host application sends a command to the module, to change the serial communication protocol from the ASCII mode to Mediatek Binary mode.

Host application initiates and carries out an EE EPO data file transfer by sending a series of MTK_BIN_EPO packet (described in paragraph 3.2 Binary Protocol) which contains 1 ~ 3 satellite data to the module. All satellite data contained in MTK_BIN_EPO packets are sourced from the EE EPO data file.

Initially, the EE EPO transfer sequence requires wait-for-ack between the sequential MTK_BIN_EPO packets. The Host application must wait for the acknowledgement before sending the next MTK_BIN_EPO packet. But as a result of improvements in the receiving side of the GNSS firmware, the wait-for-ack is no longer required by the protocol.

After all the EE EPO data is transferred, Host application sends a final MTK_BIN_EPO packet which contains the sequence number of 0xFFFF to indicate the completion of the EPO file transfer. The three satellite data fields in the MTK_BIN_EPO packet are filled with blank (0x00). If this packet is never received by the module, it has 10-seconds timeout and switch back to the default communication protocol, the ASCII mode.

On completion of the EE EPO data file transfer (indicated by the final MTK_BIN_EPO packet to the module), Host application sends a Mediatek binary command to the module, to change its serial communication protocol back to the default ASCII mode.

3.3.2.1. Error handling

If any problem occurs during the transfer process, this latter should be stopped and restarted from the beginning. Every time the protocol starts, the EPO sequence number should be reset to zero to indicate the GNSS receiver that a new transferring process has begun. The GPS receiver then needs to do preparation for the new process.

The interval of time between two continuous MTK_BIN_EPO packets must not be longer than 10 seconds. Otherwise, the GNSS receiver will determine to have problem occurred and terminate the process.

3.4. Receiver EPO Data Check

It needs to ensure that the EPO data were successfully updated into the GNSS chip. After finishing the EPO transfer protocol, make sure current data port packet format is ASCII mode.

You can query the EPO data status by issuing the PMTK_Q_EPO_INFO command:

```
$PMTK607*33<CR><LF>
```

The GNSS chip answer with a PMTK_DT_EPO_INFO packet like the following example:

```
$PMTK707,56,1468,172800,1470,151200,1468,259200,1468,259200*1F<CR><LF>
```

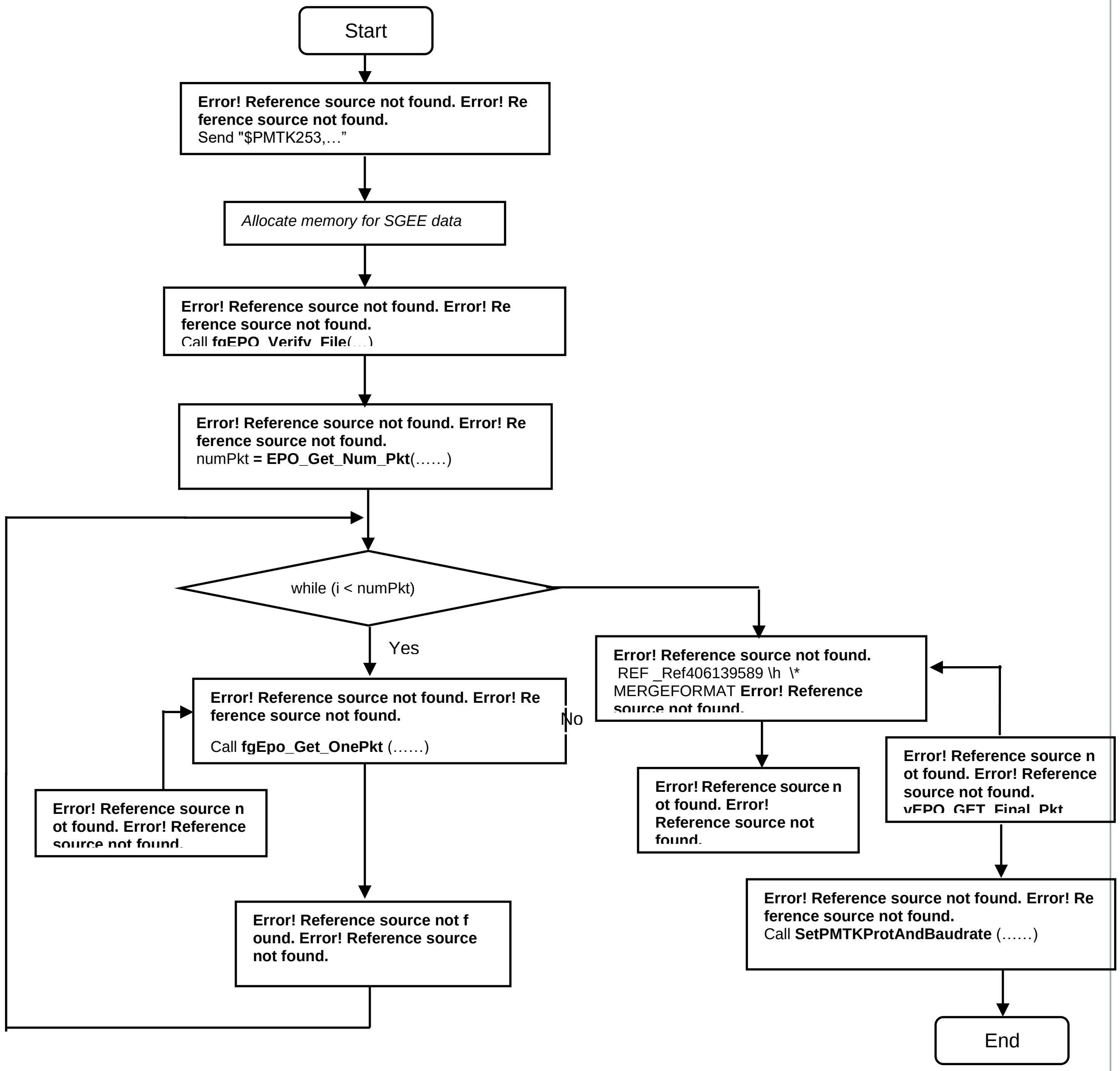
This packet shows you the information of EPO data that stored inside the GNSS receiver in terms of EPO sets uploaded, validity start Epoch, validity end Epoch and currently in use Epoch. For more details on PMTK_Q_EPO_INFO and PMTK_DT_EPO_INFO please refer to [6] V13 Software Authorized User Guide.

4. EPO TRANSFER PSEUDO CODE

4.1. Overview

This section presents pseudo code for reference purpose; this code is based on Windows platform implementation, for applications on any other platforms or environment it is programmer's responsibility to adopt, design and implement the functionalities for their specific operating environment.

4.2. Transfer Flow Chart


Start
Error! Reference source not found. Error! Reference source not found.
Send "$PMTK253,…"
Allocate memory for SGEE data
Error! Reference source not found. Error! Reference source not found.
Call fgEPO_Verify_File(…)
Error! Reference source not found. Error! Reference source not found.
numPkt = EPO_Get_Num_Pkt(……)
while (i < numPkt)
Yes
Error! Reference source not found. Error! Reference source not found.
Call fgEpo_Get_OnePkt (……)
No
Error! Reference source not found. Error! Reference source not found.
Error! Reference source not found. Error! Reference source not found.
Error! Reference source not found. REF _Ref406139589 \h * MERGEFORMAT Error! Reference source not found.
Error! Reference source not found. Error! Reference source not found.
Error! Reference source not found. Error! Reference source not found.
vEPO_GET_Final_Pkt
Error! Reference source not found. Error! Reference source not found.
Call SetPMTKProtAndBaudrate (……)
End

The above flowchart illustrates the core flow of the EE EPO data file update process as demonstrated in the pseudo code, with following notes:

- The names with bold fonts indicate they are functions
- Each block contains subsection number and its self-explanatory subtitles for what it is attempting to accomplish

- The text with Italicized fonts indicates they are general descriptions for the intended task but no reference code is provided.

4.3. Constants Definition

```
////////////////////////////////////////////////////////////////
/** \brief Definitions of constants
 *
 * #define constants
 */
////////////////////////////////////////////////////////////////
#define SUCCESS   1
#define FAIL      0

#define PREAMBLE_MTK_STX0   0x04
#define PREAMBLE_MTK_STX1   0x24
#define PREAMBLE_MTK_ETX0   0x0D
#define PREAMBLE_MTK_ETX1   0x0A

#define BYTES_PER_EPO_SEG   2304
#define NUMB_SAT_PER_PACKET    3
#define BYTES_PER_SAT         72
#define BYTES_3SATS_PER_PACKT   NUMB_SAT_PER_PACKET*BYTES_PER_SAT
#define NUMB_SAT_PER_EPO_SEG    32
#define MTKBIN_3EPO_PKT_LNG   227
```

4.4. Change Data Port to Binary Mode

At initial, the protocol setting of the communication data port is supposed to be ASCII protocol. Since, EPO data are transferred using Binary Packet, you must change the protocol setting to Binary Protocol before starting EPO Transfer Protocol.

```
///////////////////////////////////////////////////////////////
/** \fn void SetBinProt(UINT32 baud)
This function builds the PMTK command string to set the UART to the binary packet protocol
*/
//////////////////////////////////////////////////////////////
void SetBinProt(UINT32 baud)
{
   char* szCmdBuf= "$PMTK253,1,115200*00";

   OutputText(szCmdBuf);
}
```

The function OutputText(char*) must be implemented by the programmer.

4.5. Open File and Verify with Data Bytes

This function is to read the EPO file, and then verify the validity of EPO data. If the input EPO file is not in valid file length, the programmer shall terminate the process.

```
int fgEPO_Verify_File(char* epoFileName)
{
   /**
   Open the EPO file and read the file into the allocated buffer.
   This function shall return the data bytes read from the file.
   */
   // [CODE]

   /**
    * A valid data file length must be multiples of segment length.
    * GPS: the segment length is 2304 bytes
   if ((bytesRead % BYTES_PER_EPO_SEG) != 0)
   {
      bytesRead = 0;
   }

   return bytesRead;
}
```

4.6. Get Total Number of Packets

This function is to start EE EPO data file transfer protocol to send EE EPO data.

```
int EPO_Get_Num_Pkt(int recordLen)
{
   int retVal = 0;

   if (recordLen != 0)
   {
      int numbOfSet = recordLen/BYTES_PER_EPO_SEG;
      retVal = (numbOfSet*NUMB_SAT_PER_EPO_SEG)/NUMB_SAT_PER_PACKET;
   }
   return retVal;
}
```

4.7. Start EPO Download Function

Now the data in the data port will be viewed as Binary Packet format. Please create a thread to transmit / receive binary packets for the data port.

```
{
   ……

   /** Allocate buffer for the EPO data
   // This must be implemented by the programmer.
   // ……



   /**
     * Call EPO download engine, with passing parameters
       - pEPOBuffer: EPO data buffer
       - fSize: EPO data size
       - numPkts: number og packets to send
       - nBaudRate: the current baud rate
    */
    EMOD_DLEng(nBaudRate, numPkts, fSize, pEPOBuff);
}
```

Please note that the code here is for reference and is based on the Windows platform only, as it depends on the programmer’s design as how to use a thread or task to run an EE EPO update process, or if to use one after all.

4.8. Get Packets and Send

This function starts EE-EPO data transfer protocol to send EE-EPO data to the receiver.

```
bool EMOD_DLEng(UINT32 baud, int numPkt, DWORD epoSize, const BYTE* pEPOBuff)
{
   for (int i = 0; i < numPkt; i++)
   {
      if (fgEpo_Get_OnePkt(dataBuf, sizeof(dataBuf), epoSeq, pEPOBuff) == SUCCESS)
      {
         //* Send the current packet
         OutputBytes(dataBuf , MTKBIN_3EPO_PKT_LNG);

         //* Update the last Epo SEQ number
         lastEpoSeq = epoSeq;
         epoSeq++;
         ::Sleep(100);
```

```
        }
    }
}
```

4.9. Build Data Packets

This function, fgEPO_Get_One_Pkt(…), takes out three SAT data from the EE EPO data file and encapsulate them in a MTK_BIN_EPO packet with appropriate EPO SEQ number.

```
int fgEpo_Get_OnePkt(unsigned int seq, BYTE* pEpoRecord, BYTE* pData, int
size)
{
    int retVal = FAIL;

    /**
     * Sanity check the input argument
     */
    // [CODE]

    if ((seq+1)*BYTES_3SATS_PER_PACKT <= nEpoRecordSize)
    {
        int indx = 0;

        pData[indx++] = PREAMBLE_MTK_STX0;
        pData[indx++] = PREAMBLE_MTK_STX1;

        pData[indx++] = 0xE3;
        pData[indx++] = 0x0;

        pData[indx++] = 0xD3;
        pData[indx++] = 0x02;

        pData[indx++] = seq & 0xFF;
        pData[indx++] = (seq >> 8) & 0xFF;

        memcpy((BYTE*)&(pData[indx]),
(BYTE*)&(pEpoRecord[seq*BYTES_3SATS_PER_PACKT]), BYTES_3SATS_PER_PACKT);

        indx += BYTES_3SATS_PER_PACKT;

        BYTE chksum = ComputeChkSum(pData+2, 6 + BYTES_3SATS_PER_PACKT);
//* len byte + Cmd byte + Seq byte = 6
        pData[indx++] = chksum;
```

```
      pData[indx++] = PREAMBLE_MTK_ETX0;
      pData[indx++] = PREAMBLE_MTK_ETX1;

      retVal = SUCCESS;
   }
```

fgEPO_Get_One_Pkt also contains the code to fill the MTK_BIN_EPO packet when there are less than three (3) sat data left from the data buffer.

```
int fgEpo_Get_OnePkt(unsigned int seq, BYTE* pEpoRecord, BYTE* pData, int
size)
{
   /**
    * Sanity check the input argument
    */
   // [CODE]

   if ((seq+1)*BYTES_3SATS_PER_PACKT <= nEpoRecordSize)
   {
   }
   else
   {
      int trailBytes = nEpoRecordSize - seq*BYTES_3SATS_PER_PACKT;

      if ((trailBytes % 72) == 0)
      {
         int indx = 0;
         BYTE fillBytes[BYTES_3SATS_PER_PACKT];   // To prefill the bytes
with numerial 0
         memset (fillBytes, 0x30, sizeof(fillBytes));
         memcpy((BYTE*)&(fillBytes[0]),
(BYTE*)&(pEpoRecord[seq*BYTES_3SATS_PER_PACKT]), trailBytes);


         pData[indx++] = PREAMBLE_MTK_STX0;
         pData[indx++] = PREAMBLE_MTK_STX1;

         pData[indx++] = 0xE3;
         pData[indx++] = 0x0;

         pData[indx++] = 0xD3;
         pData[indx++] = 0x02;
         pData[indx++] = seq & 0xFF;
         pData[indx++] = (seq >> 8) & 0xFF;

         memcpy((BYTE*)&(pData[indx]), (BYTE*)&fillBytes[0],
BYTES_3SATS_PER_PACKT);

         indx += BYTES_3SATS_PER_PACKT;

         BYTE chksum = ComputeChkSum(pData+2, 6 + BYTES_3SATS_PER_PACKT);
//* len byte + Cmd byte + Seq byte = 6
         pData[indx++] = chksum;

         pData[indx++] = PREAMBLE_MTK_ETX0;
         pData[indx++] = PREAMBLE_MTK_ETX1;

         retVal = SUCCESS;
      }
   }

   return retVal;
}
```

Send current MTK_BIN_EPO packet. The packet size of MTK_BIN_EPO is MTKBIN_3EPO_PKT_LNG.

The call to OutputBytes() must be made by the programmer.

4.10. Build Final Data Packet

Generate final MTK_BIN_EPO packet to indicate the GPS receiver that the process is finish.

```
int vEPO_GET_Final_Pkt(BYTE* pData, int size)
{
   int retVal = FAIL;
   BYTE val = 0;

   /**
    * Sanity check the input argument
    */
   // [CODE]

   int indx = 0;

   pData[indx++] = PREAMBLE_MTK_STX0;
   pData[indx++] = PREAMBLE_MTK_STX1;
```

```
    pData[indx++] = 0xE3;
    pData[indx++] = 0x0;

    pData[indx++] = 0xD3;
    pData[indx++] = 0x02;

    pData[indx++] = 0xFF;
    pData[indx++] = 0xFF;

    memcpy((BYTE*)&(pData[indx]), &val, BYTES_3SATS_PER_PACKT);

    indx += BYTES_3SATS_PER_PACKT;

    BYTE chksum = ComputeChkSum(pData+2, 6 + BYTES_3SATS_PER_PACKT);
    pData[indx++] = chksum;

    pData[indx++] = PREAMBLE_MTK_ETX0;
    pData[indx++] = PREAMBLE_MTK_ETX1;
}
```

Like before, call the SendData function to send the final MTK_BIN_EPO packet to the serial port.

The call to SendData() must be made by the programmer.

4.11. Build Packet and Change Port to ASCII Mode

Switch UART protocol setting to ASCII mode and baud rate 115200.

```
void SetTextProt(UINT32 baud)
{
    int indx = 0;
    BYTE cmd[200] = { 0 };

    cmd[indx++] = 0x04; //* Preamble
    cmd[indx++] = 0x24;

    cmd[indx++] = 0x0E; //* Len 2 bytes for MTK_BIN_EPO packet
    cmd[indx++] = 0x00;

    cmd[indx++] = 0xFD; //* Command ID for MTK_BIN_EPO packet
    cmd[indx++] = 0x00;

    cmd[indx++] = PCKT253_SETPMTKPROTO_FLAG;  //* PMTK protocol
```

```
   cmd[indx++] = (baud >> 24) & 0x000000FF;
   cmd[indx++] = (baud >> 8) & 0x000000FF;
   cmd[indx++] = (baud >> 16) & 0x000000FF;
   cmd[indx++] = baud & 0x000000FF;
   cmd[indx++] = ComputeChkSum(&cmd[2], indx-2);

   cmd[indx++] = 0x0D;
   cmd[indx++] = 0x0A;

   OutputBytes(cmd, indx);
}
```

The call to OutputBytes() must be made by the programmer.

5. EPO TRANSFER APPLICATION PACKAGE

TDepo is a console application for Windows, this tool illustrates a sample implementation of the Mediatek's EPO update protocol.

TDepo is developed and intended to run in the following environment:

- Windows 7 or later
- Visual Studio 2010
- .NET 4.0

5.1. TDepo Usage

5.1.1. TDepo Directory and Files

In the simplest setup, create a folder directory "TDepo" and copy the following files into the directory:

1. **TDepo.exe:** The Executable
2. **TDepoCfg.txt:** Configuration file to provide parameters for running the application. This file is highly recommended, unless user choose to provide the parameters through command line. This file is assumed to reside in the same directory as the executable.
3. **MTK30.EPO:** The EPO data file for MediaTek's GPS only EPO. This file can be stored in any of the user's directory – provided that its full path is included wither from the command line argument or in the configuration file TDepoCfg.txt.

Below is an example of the contents of the TDepoCfg.txt file:

```
-pCOM26
-b9600
-eC:\TDepo\MTK30.EPO
```

The parameters contained in the configuration file can also be entered from command line.

5.1.2. EPO File Source

There is significant information available pertaining where to get and how to use the EPO file. But for the purpose of this application note and the nature that is to introduce the EPO download sample application, it is suffice to say that the EPO file is available for download at Telit's EPO server.

There are credentials (user name, password, and so on) that are required in logging. Please contact Telit customer support for more details about the information.

The EPO file can be either downloaded through an internet browser, or, through the TelitView application. The "EPO Host Manager" window in the TelitView application provides the access and functionality to download the EPO file.

Please take a note about the directory where the EPO file is stored at after download from the server and provide the full path either in the TDepo command line or use the configuration file.

5.1.3. Application Launch

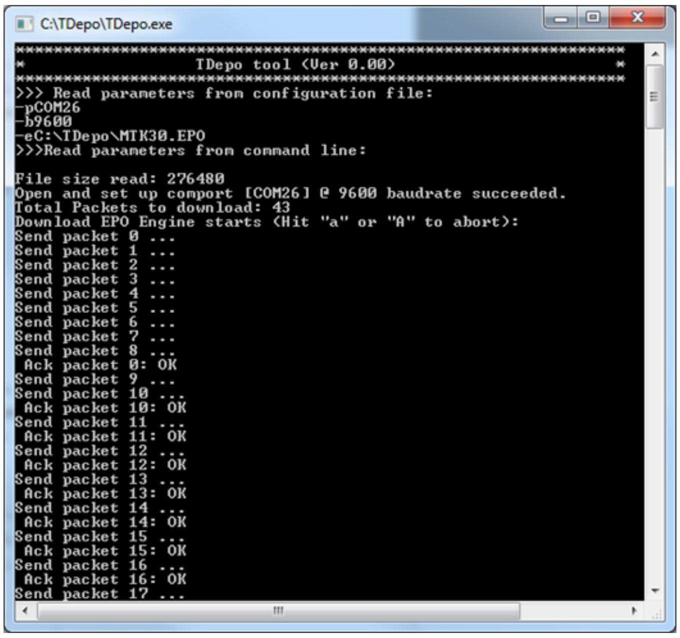
From command line, type “TDepo –h”, the application launch displays the following message:

```
*****************************************************************
*                         TDepo tool (Ver 0.00)                 *
*****************************************************************
>>> Help menu:
TDepoCfg -p<port name> -b<baud rate> -e<EPO filepath>
  <port name>: PC com port name (COM2, COM39, etc)
  <baud rate>: Init baud rate (115200, 9600, etc).
          This is the baud rate that communication is established on
          before the EPO download, and return to after the EPO download.
  <EPO filepath>: complete file path for the EPO file file
TDepoCfg -h<no parameter>
  : help menu (this menu)
  - Not parameter-order sensative
  - Command line inputs take priority over parameters from file

Press any key to continue . . .
```

Use parameter “-h” from the command line to display the menu printout on the screen.

User is expected to provide other supported argument except “-h”, or no argument at all but provide the necessary configuration file for the application to run.



```
*****************************************************************
*                         TDepo tool (Ver 0.00)                 *
*****************************************************************
>>> Read parameters from configuration file:
-pCOM26
-b9600
-eC:\TDepo\MTK30.EPO
>>>Read parameters from command line:

File size read: 276480
Open and set up comport [COM26] @ 9600 baudrate succeeded.
Total Packets to download: 43
Download EPO Engine starts (Hit "a" or "A" to abort):
Send packet 0 ...
Send packet 1 ...
Send packet 2 ...
Send packet 3 ...
Send packet 4 ...
Send packet 5 ...
Send packet 6 ...
Send packet 7 ...
Send packet 8 ...
 Ack packet 0: OK
Send packet 9 ...
Send packet 10 ...
 Ack packet 10: OK
Send packet 11 ...
 Ack packet 11: OK
Send packet 12 ...
 Ack packet 12: OK
Send packet 13 ...
 Ack packet 13: OK
Send packet 14 ...
 Ack packet 14: OK
Send packet 15 ...
 Ack packet 15: OK
Send packet 16 ...
 Ack packet 16: OK
Send packet 17 ...
```

Figure 5-1 Example of Application execution

5.1.4. TTFF with EPO Available in GPS

On completion of update EPO to the GNSS module, user may launch TelitView, or PowerGPS tool (a tool provided by MediaTek) and command a reset.

The most evident example would be the improvement of TTFF on a Warm start or Cold start. With the EPO now is available in the GNSS module, a warm start of GPS device would yield a nominal TTFF of 1 – 3 seconds.

5.2. TDepo Source Code

The table below lists and describes the TDepo source files.

Files	Description
TDepo.h **TDepo.cpp**	Application’s main file with the main function, command line parser, open com port, and other set-up functions.
Com_port.h **Com_port.cpp**	The module to handle the com port and serial communication.
HostEpoEng.h **HostEpoEng.cpp**	The Host EPO update engine that executes the EPO update process based on the protocol.
Messages.h **Messages.cpp**	The functions that provide the implementation of some functions that are not a part of the EPO update, but nevertheless necessary to be included to support the Mediatek interface.

Table 5-1 TDepo Source Code Files

For more information, please refer to the source code listing for implementation details.

6. ACRONYMS

	Description
TTSC	Telit Technical Support Centre
USB	Universal Serial Bus
EE	Extended Ephemeris
EPO	Extended Prediction Orbit
OEM	Original Equipment Manufacturer
ASCII	American Standard Code for Information Interchange
BE	Broadcast Ephemeris
NMEA	National Marine Electronics Association
UART	Universal Asynchronous Receiver Transmitter
SRAM	Static Random Access Memory
UTC	Co-ordinated Universal Time
RTC	Real Time Clock
TTFF	Time to First Fix

7. DOCUMENT HISTORY

Revision	Date	Changes
0	2020-09-16	First issue

SUPPORT INQUIRIES

Link to **www.telit.com** and contact our technical support team for any questions related to technical issues.

www.telit.com

Telit Communications S.p.A.
Via Stazione di Prosecco, 5/B
I-34010 Sgonico (Trieste), Italy

Telit IoT Platforms LLC
5300 Broken Sound Blvd, Suite 150
Boca Raton, FL 33487, USA

Telit Wireless Solutions Inc.
3131 RDU Center Drive, Suite 135
Morrisville, NC 27560, USA

Telit Wireless Solutions Co., Ltd.
8th Fl., Shinyoung Securities Bld.
6, Gukjegeumyung-ro8-gil, Yeongdeungpo-gu
Seoul, 150-884, Korea

Telit Wireless Solutions Ltd.
10 Habarzel St.
Tel Aviv 69710, Israel

Telit Wireless Solutions
Technologia e Servicos Ltda
Avenida Paulista, 1776, Room 10.C
01310-921 São Paulo, Brazil