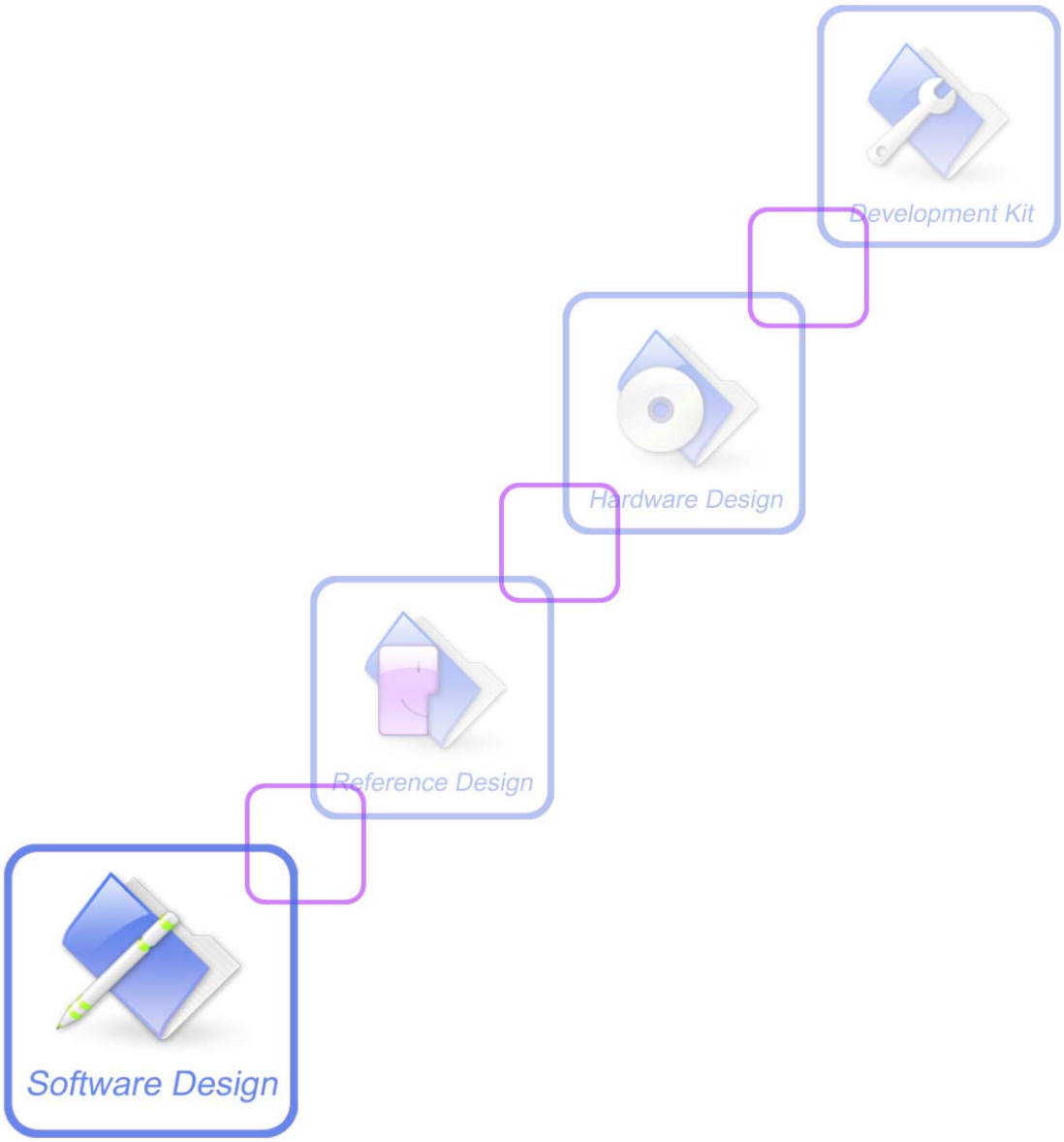
A company of SIM Tech

SIM28/68R/68V EPO-II Protocol



Document Title:	SIM28 / 68R / 68V EPO-II Protocol
Version:	V1.00
Date:	2013-01-07
Status:	Release
Document Control ID:	SIM28 / 68R / 68V EPO-II_Protocol_V1.00

General Notes

SIMCOM offers this information as a service to its customers, to support application and engineering efforts that use the products designed by SIMCOM. The information provided is based upon requirements specifically provided to SIMCOM by the customers. SIMCOM has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by SIMCOM within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

Copyright

Version History

Version	Chapter	What is new
V1.00	Original version	Original

Contents

Tables

Figures

1 Introduction

1.1 Scope of the document

This document presents details EPO file format and transferring protocol for EPO Management Tool development.

1.2 Related documents

(1). EPO-II Format and Protocol (Release Date:2011-04-25, Mediatek Inc.)

1.3 Term abbreviation

Table 1-1: Term abbreviation

Term	Definition
EPO	Extended Prediction Orbit
GPS	Global Positioning System
FTP	File Transfer Protocol

2 MTK Binary Protocol

This document introduce messages in MTK Binary Protocol format.

Table 2-1: MTK Binary Protocol format

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

Table 2-2: MTK Binary Protocol segments description

Parameter	Length (Byte)	Contents
Preamble	2	0x2404
Length	2	Total number bytes in the packet from Preamble to End Word. Maximum packet size: 256 bytes Use little endian Use one byte alignment
Command ID	2	(1) 0 ~ 999: conform to PMTK ASCII Protocol (2) 1000 ~ 65535: designated for MTK Binary Protocol
Data	Variable	Data to be transfered
Checksum	1	The checksum is the 8-bit exclusive OR of all bytes in the packet between but not including the “Preamble” and the “Checksum”
End Word	2	0x0A0D

Please refer to section 7.4 for the samples of MTK binary packet types.

3 EPO File Format

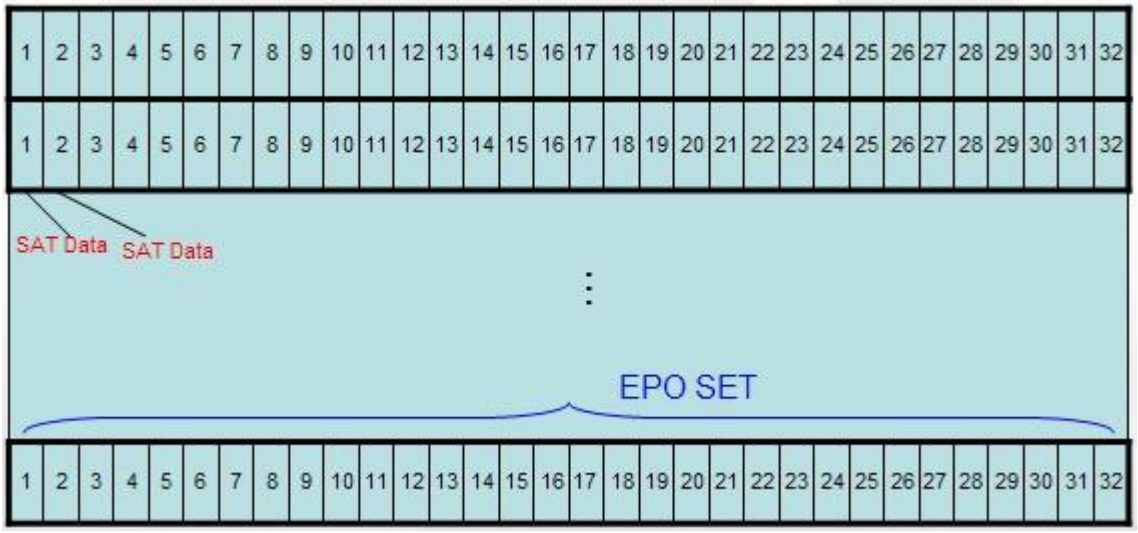


Figure 3-1: EPO file Format

The basic unit of an EPO file is GPS satellite data, the data size of a GPS satellite data is 72 bytes.

One EPO SET contains 32 GPS satellites data.

The data size for an EPO SET is 2304 bytes(72 x 32).

Each EPO file contains several EPO SETs. The file size must be a multiple of 2304.

An EPO SET is valid for 6 hours. Therefore, there will be 4 EPO SETs for one day(2304x4= 9216bytes).

Client can get EPO file from MTK or SIMCOM FTP or HTTP site.

Note:

1. ***Our HTTP address is: http://wm.sim.com/MTK30.EPO***
2. ***Our FTP address is: ftp://116.228.221.52***
   User name: customer
   Password : 111111

4 EPO Binary Packet Format

For the convenience, we named the EPO binary packet as MTK_BIN_EPO packet (packet type 723).

Table 4-1: EPO Binary Packet Format

Preamble		Length	Command ID	Data				Checksum	End Word	
0x04	0x24	0x00E3	0X02D3	EPO SEQ	SAT Data	SAT Data	SAT Data	0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	2 Bytes	72 Bytes	72 Bytes	72 Bytes	1 Byte	2 Bytes	

An EPO file will be divided into several SAT Data and encapsulated in several MTK_BIN_EPO packets to be transferred to MTK GPS receiver. Each MTK_BIN_EPO packet contains a 2-byte EPO SEQ and 3 SAT Data. The packet length of MTK_BIN_EPO is 227 bytes. The EPO SEQ is used for synchronization of MTK_BIN_EPO packets in transferring protocol.

Sometimes, there's no enough EPO data to full fill the three SAT Data fields. You can leave some of the three fields as blank, that is, to fill them with 0x00. A MTK_BIN_EPO packet that only contains 0 ~ 2 SAT Data is possible and acceptable. The following three MTK_BIN_EPO packets are examples:

Table 4-2: MTK_BIN_EPO (contains 2 SAT Data)

Preamble		Length	Command ID	Data				Checksum	End Word	
0x04	0x24	0x00E3	0X02D3	EPO SEQ	SAT Data	SAT Data	0x00	0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	2 Bytes	72 Bytes	72 Bytes	72 Bytes	1 Byte	2 Bytes	

Table 4-3: MTK_BIN_EPO (contains 1 SAT Data)

Preamble		Length	Command ID	Data				Checksum	End Word	
0x04	0x24	0x00E3	0X02D3	EPO SEQ	SAT Data	0x00	0x00	0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	2 Bytes	72 Bytes	72 Bytes	72 Bytes	1 Byte	2 Bytes	

Table 4-4: MTK_BIN_EPO (contains no SAT Data)

Preamble		Length	Command ID	Data				Checksum	End Word	
0x04	0x24	0x00E3	0X02D3	EPO SEQ	0x00	0x00	0x00	0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	2 Bytes	72 Bytes	72 Bytes	72 Bytes	1 Byte	2 Bytes	

MTK GPS receiver will return an acknowledge packet for each received MTK_BIN_EPO. For convenience, we named the acknowledge packet as MTK_BIN_ACK_EPO (packet type 2)

Table 4-5: Acknowledge for MTK_EPO_BIN

Preamble		Length	Command ID	Data		Checksum	End Word	
0x04	0x24	0x000C	0X0002	EPO SEQ	Result	0xHH	0x0D	0x0A
2 Bytes		2 Bytes	2 Bytes	2 Bytes	1 Bytes	1 Byte	2 Bytes	

EPO SEQ: sequence number to indicate the corresponding received MTK_BIN_EPO

Result: '0' the received MTK_BIN_EPO is invalid and means fail

'1' the received MTK_BIN_EPO is valid and means success.

5 EPO Data Transfer

EPO data are packeted in MTK_BIN_EPO packets using MTK Binary Protocol and then be transferred from EPO Management Tool to EPO Transfer Agent in a MTK GPS receiver.
At the beginning of the transferring protocol, EPO Management Tool have to split the EPO file and encapsulated into several MTK_BIN_EPO packets, and gives each MTK_BIN_EPO packet a sequence number starting from zero. The sequence number is in order to make sure the MTK_BIN_EPO packets are transferred in correct order and there are no packet miss. Then EPO Management Tool and EPO Transfer Agent follow the “EPO Data Transfer Protocol” to transfer EPO data into MTK GPS receiver.

(1). EPO Management Tool: Send one MTK_BIN_EPO packet, which contains 1 ~3 SAT data, to MTK GPS receiver. The sequence number in the packet starts from zero and will be added one for each of the following MTK_BIN_EPO packets.

(2). EPO Management Tool: Wait for the MTK_BIN_ACK_EPO that has the same sequence number with the transmitted MTK_BIN_EPO.

(3). EPO Transfer Agent in MTK GPS receiver: Receive MTK_BIN_EPO packet from EPO Management Tool. Then verify the validity of EPO data in the packet. If it’s a correct packet, Transfer Agent will return a MTK_BIN_ACK_EPO packet to indicate success; otherwise, return a MTK_ BIN_ACK_EPO packet to indicate fail.

(4). EPO Transfer Agent in MTK GPS receiver: Wait for the next EPO packet

(5). EPO Management Tool: Receive MTK_ BIN_ACK_EPO packet. If the acknowledge indicates success, Management Tool prepares to send next MTK_BIN_EPO packet; otherwise, something wrong and needs to exit the protocol.

(6). Repeat steps (1) ~ (5) until all the EPO data are transferred.

(7). EPO Management Tool: Send a final MTK_BIN_EPO packet which contains sequence number of 0xFFFF to indicate the finish of the protocol. The 3 SAT data fields in the final MTK_BIN_EPO packet are all blank.

Figure 5-1: EPO Data Transfer procedure

5.2. Error Handling

If there is any problem occurs in the transferring protocol, you shall stop the process and restart the transferring protocol again. Every time when the protocol starts, the EPO sequence number should be reset to zero to indicate the GPS 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 shall not be longer than 10 seconds. Otherwise, the GPS receiver will determine to have problem occurred and terminate the process.

5.3. Check EPO data in GPS chip

It needs to ensure that the EPO data were successfully updated into the GPS chip. After finishing the EPO transfer protocol, make sure current UART packet format is NMEA mode. Then you can issue the PMTK_Q_EPO_INFO command

$PMTK607*33<CR><LF>

to query the EPO data status. The GPS chip will return you PMTK_DT_EPO_INFO like below

$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 GPS chip. For 14-day EPO file, the first argument following PMTK707 will be 56; for 7-day EPO file, it will be 28; (1468, 172800) means the starting GPS time (GPS week, GPS TOW) of the EPO data, and (1470, 151200) means the ending GPS time (GPS week, GPS TOW) of the EPO data. You have to convert (GPS week, GPS TOW) into UTC time format, so as to compare the UTC time to verify that the EPO data stored in the flash matches that of the EPO file.

Please refer to section 7 for the details of EPO related PMTK commands: PMTK_Q_EPO_INFO, PMTK_DT_EPO_INFO

6 Pseudo code for EPO transferring process

```
00001: #define MTKBIN_3EPO_PKT_LNG 227
00002: // At initial, the protocol setting of the communication UART is supposed to be
00003: // PMTK protocol.
00004: // Since EPO data are transferred using MTK Binary Packet, you have to change
00005: // the protocol setting to MTK Binary Protocol
00006: // before starting EPO Transfer Protocol. You can use PMTK command 253 to change
00007: // the UART protocol setting.
00008: // Please refer to 7.1 for the details of PMTK command 253.
00009: // The function SendPmtkCmd must be implemented by the programmer.
00010: // NOTE: we suggest the user to explicitly specify baudrate when changing UART
00011: // packet protocol, for example,
00012: // $PMTK253,1,115200*00<CR><LF>
00013: SendPmtkCmd( $PMTK253,1,0*37<CR><LF> );
00014: // Now the data in the UART will be viewed as MTK Binary Packet format. Please
00015: // create a thread to transmit / receive MTK
00016: // binary packets for the UART.
00017: // The thread TMtkBinCmdThread must be implemented by the programmer.
00018: pMtkBinCmdThread = new TMtkBinCmdThread();
00019: // Read in the EPO file, and then verify the validity of EPO data. If the input
00020: // EPO file is not in valid MTK EPO format, the
00021: // programmer shall terminate the process.
00022: // Please check the file size is a multiple of 2304.
00023: // The function fgEPO_Verify_File must be implemented by the programmer.
00024: if (!fgEPO_Verify_File(pEpoFile))
00025: return;
00026: // Get total number of MTK_BIN_EPO packets that will be sent.
00027: // Total number = ceil ((file size / 2304) / 3)
00028: // The function i2EPO_Get_Num_Pkt must be implemented by the programmer.
00029: i4NumSvEpoPkt = i2EPO_Get_Num_Pkt(pEpoFile);
00030: // Start EPO Data Transfer Protocol to send EPO data
00031: u2EpoSeq = 0;
00032: u2LastEpoSeq = 0;
00033: for (i = 0; i < i4NumSvEpoPkt; i++)
00034: {
00035:         // fgEPO_Get_One_Pkt takes out three SAT data from the EPO file and encapsulated
00036:         // them in a MTK_BIN_EPO packet
00037:         // with appropriate EPO SEQ number.
00038:         // In order to save the total transferring time, we suggest to generate current
00039:         // EPO packet first, and then wait for
00040:         // MTK_BIN_ACK_EPO acknowledge of the previous MTK_BIN_EPO packet from the
00041:         // GPS receiver.
00042:         // The function fgEPO_Get_One_Pkt must be implemented by the programmer.
00043:         if (fgEPO_Get_One_Pkt(u2EpoSeq, pEpoFile, szPktData))
00044:         {
00045:                 // Wait for EPO acknowledge. The GPS receiver will return a MTK_BIN_ACK_EPO
00046:                 // acknowledge packet after
00047:                 // receiving and processing previous MTK_BIN_EPO packet. Please refer to
00048:                 // section 7.4 for the details of
00049:                 // MTK_BIN_ACK_EPO acknowledge packet.
00050:                 // If the acknowledge indicates failure, you shall terminate the process.
00051:                 // The function fgWait_Epo_Ack must be implemented by the programmer.
00052:                 if (!fgWait_Epo_Ack(u2LastEpoSeq))
00053:                 { return; }
```

```
00054:                      // Send current MTK_BIN_EPO packet. The packet size of MTK_BIN_EPO is
00055:                      // MTKBIN_3EPO_PKT_LNG..
00056:                      // The function SendData must be implemented by the programmer.
00057:                      pPortMtkBinThread->SendData(szPktData, MTKBIN_3EPO_PKT_LNG);
00058:                      // Update sequence number
00059:                      u2LastEpoSeq = u2EpoSeq;
00060:                      u2EpoSeq++;
00061:                  }
00062: }
00063: // Generate final MTK_BIN_EPO packet to indicate the GPS receiver that the process
00064: // is finish.
00065: // The function fgEPO_Get_Final_Pkt must be implemented by the programmer.
00066: vEPO_Get_Final_Pkt(szPktData);
00067: // Send final MTK_BIN_EPO packet to the GPS receiver. The packet size of
00068: // MTK_BIN_EPO is MTKBIN_3EPO_PKT_LNG.
00069: // Then the process is finished.
00070: // The function SendData must be implemented by the programmer.
00071: pPortMtkBinThread->SendData(szPktData, MTKBIN_3EPO_PKT_LNG);
00072: // Switch UART protocol setting to PMTK packet format and baudrate 115200 for
00073: // the communication UART.
00074: // Please refer to section 7.2 for the details of "Change UART format packet".
00075: // The function SendMtkBinCmd must be implemented by the programmer.
00076: SendMtkBinCmd(0x04 0x24 0x0E 0x00 0xFD 0x00 0x00 0x00 0xC2 0x01 0x00 0x30 0x0D 0x0A);
```

7 EPO Related PMTK Commands

7.1 Packet Type: 253 PMTK_SET_OUTPUT_FMT

This command set data output format and baudrate for current port

Table 7-1: 253 PMTK_SET_OUTPUT_FMT

DataField: PMTK253,Flag,Baudrate			
Example: Switch the UART protocol format to BINARY mode, and use default baudrate 115200 $PMTK253,1,0*37<CR><LF> Switch the UART protocol format to NMEA mode, and use baudrate 9600 $PMTK253,0,9600*09<CR><LF>			
Name	**Unit**	**Default**	**Description**
Flag	--	0	0 - NMEA mode 1 - binary mode
Baudrate	--	115200	baudrate for the new output mode 0: use default baudrate (do not suggest to use this) UART1: default baudrate will be the value set in “Data Port UART1 Baudrate” of Corebuilder. UART0: default baudrate will be the value set in “NMEA Baudrate” of

			Corebuilder We highly suggest you specifying an explicit baudrate value, all the other possible values are 4800, 9600, 14400, 19200, 38400, 57600, 115200.

Note:

When you switch from binary mode to NMEA mode, you will receive a binary ACK (Packet Type 1) after the command is processed. Please refer to MTK Binary Packet Types for the detail of ACK packet. When you switch from NMEA mode to binary mode, NO ACK will be sent.

7.2 Packet Type: 607 PMTK_Q_EPO_INFO

This command query the EPO data status stored in the GPS chip

Table 7-2: 607 PMTK_Q_EPO_INFO

DataField: PMTK607			
Example: $PMTK607*33<CR><LF>			
Name	**Unit**	**Defalult**	**Description**
--	--	--	--

The response of this command is PMTK_DT_EPO_INFO packet. See chapter below.

7.3 Packet Type: 707 PMTK_DT_EPO_INFO

This response packet contains EPO data status stored in GPS chip.

Table 7-3: 707 PMTK_DT_EPO_INFO

DataField: PMTK707,Set,FWN,FTOW,LWN,LTOW,FCWN,FCTOW,LCWN,LCTOW			
Example: $PMTK707,56,1468,172800,1470,151200,1468,259200,1468,259200*1F<CR><LF>			
Name	**Unit**	**Defalult**	**Description**
Set	--	--	Total number sets of EPO data stored in chip
FWN	--	--	GPS week number of the first set of EPO data stored in chip respectively
FTOW	--	--	GPS week TOW of the first set of EPO data stored in chip respectively
LWN	--	--	GPS week number of the last set of EPO data stored in chip respectively
LTOW	--	--	GPS week TOW of the last set of EPO data stored in chip respectively
FCWN	--	--	GPS week number of the first set of EPO data that are currently used respectively
FCTOW	--	--	GPS week TOW of the first set of EPO data that are currently used respectively
LCWN	--	--	GPS week number of the last set of EPO data that are currently used respectively
LCTOW	--	--	GPS week TOW of the last set of EPO data that are currently used respectively

7.4 EPO Related MTK Binary Packet Type

7.4.1 Acknowledge Packet (Packet Type 1)

This packet is usually returned after receiving a MTK binary packet.

Table 7-4: MTK Binary Acknowledge Packet (Packet Type 1) Example

DataField: <packet type><Flag>			
Example: Receive a valid MTK binary packet and return a success flag 0x04 0x24 0x0C 0x00 0x01 0x00 0xFD 0x00 0x03 0xF3 0x0D 0x0A			
Parameter	**Length(Byte)**	**Example**	**Contents**
Preamble	2	0x2404	
Length	2	0x000C	Length is 12 bytes
Command ID	2	0x0001	EPO acknowledge packet
packet type	2	0x00FD	The acknowledge is responding to packet type 253
Flag	1	0x03	0x00, 0x01: invalid packet 0x02: fail 0x03 success
Checksum	1	0xF3	The checksum is the 8-bit exclusive OR of all bytes in the packet between but not including the “Preamble” and the “Checksum”
End Word	2	0x0A0D	

7.4.2 EPO Acknowledge Packet (Packet Type 2)

This packet is usually returned after receiving an EPO binary packet.

Table 7-5: MTK Binary Acknowledge Packet(Packet Type 2) Example1

DataField: < Acknowledge Info >			
Example: Receive a valid EPO packet for sequence number 0x56 0x04 0x24 0x0C 0x00 0x02 0x00 0x56 0x00 0x01 0x59 0x0D 0x0A			
Parameter	**Length(Byte)**	**Example**	**Contents**
Preamble	2	0x2404	
Length	2	0x000C	Length is 12 bytes
Command ID	2	0x0002	EPO acknowledge packet
Acknowledge Info	3	0x56 0x00 0x01	Valid EPO packet for sequence 0x56
Checksum	1	0x59	The checksum is the 8-bit exclusive OR of all bytes in

			the packet between but not including the “Preamble” and the “Checksum”
End Word	2	0x0A0D	

Table 7-6: MTK Binary Acknowledge Packet(Packet Type 2) Example2

DataField: < Acknowledge Info >			
Example: Receive an invalid EPO packet for sequence number 0x56 0x04 0x24 0x0C 0x00 0x02 0x00 0x56 0x00 0x00 0x58 0x0D 0x0A			
Parameter	**Length(Byte)**	**Example**	**Contents**
Preamble	2	0x2404	
Length	2	0x000C	Length is 12 bytes
Command ID	2	0x0002	EPO acknowledge packet
Acknowledge Info	3	0x56 0x00 0x00	invalid EPO packet for sequence 0x56
Checksum	1	0x58	The checksum is the 8-bit exclusive OR of all bytes in the packet between but not including the “Preamble” and the “Checksum”
End Word	2	0x0A0D	

7.2.3. Change UART Format Packet (Packet Type 253):

To change UART communication protocol and baudrate.

Table 7-7: MTK Binary Change UART Format Packet (Packet Type 253) Example1

DataField: < protocol><UART baudrate >			
Example: Change UART to PMTK protocol and baudrate 115200 0x04 0x24 0x0E 0x00 0xFD 0x00 0x00 0x00 0xC2 0x01 0x00 0x30 0x0D 0x0A			
Parameter	**Length(Byte)**	**Example**	**Contents**
Preamble	2	0x2404	
Length	2	0x000E	Length is 14 bytes
Command ID	2	0x00FD	Change UART packet protocol
protocol	1	0x00	PMTK protocol
UART baudrate	4	0x00 0xC2 0x01 0x00	UART baudrate 115200
Checksum	1	0x30	The checksum is the 8-bit exclusive OR of all bytes in the packet between but not including the “Preamble” and the “Checksum”
End Word	2	0x0A0D	

Table 7-8: MTK Binary Change UART Format Packet (Packet Type 253) Example2

DataField: < protocol><UART baudrate >			
Example: Change UART to PMTK protocol and use default baudrate 0x04 0x24 0x0E 0x00 0xFD 0x00 0x00 0x00 0x00 0x00 0x00 0xF3 0x0D 0x0A			
Parameter	**Length(Byte)**	**Example**	**Contents**
Preamble	2	0x2404	
Length	2	0x000E	Length is 14 bytes
Command ID	2	0x00FD	Change UART packet protocol
protocol	1	0x00	PMTK protocol
UART baudrate	4	0x00 0x00 0x00 0x00	UART baudrate use default value
Checksum	1	0xF3	
End Word	2	0x0A0D	

Contact us:

Shanghai SIMCom wireless solutions Ltd.

Address: Building A, SIM Technology Building, No. 633 Jinzhong Road, Shanghai,
P. R. China 200335

Tel: +86 21 3252 3300

Fax: +86 21 3252 2030

URL: www.sim.com/wm