11 Appendix B. NMEA Message Format

The GPS receiver outputs data in NMEA-0183 format at 9600 Baud, 8 bits, no parity bit, and 1 stop bit. The GGA and RMC message packet formats are explained in this section.

11.1 GGA - GPS fix data

Time and position, together with GPS fixing related data (number of satellites in use, and the resulting HDOP, age of differential data if in use, etc.).

`$GPGGA,hhmmss.ss,Latitude,N,Longitude,E,FS,NoSV,HDOP,msl,m,Altref,m,DiffAge,DiffStation*cs<CR><LF>`

Name	ASCII String		Description
	Format	Example	
$GPGGA	string	$GPGGA	Message ID: GGA protocol header
hhmmss.ss	hhmmss.sss	092725.00	UTC Time: Current time
Latitude	dddmm.mmmm	4717.11399	Latitude: Degrees + minutes
N	character	N	N/S Indicator: N=north or S=south
Longitude	dddmm.mmmm	00833.91590	Longitude: Degrees + minutes
E	character	E	E/W indicator: E=east or W=west
FS	1 digit	1	Position Fix Indicator (See Table below)
NoSV	numeric	8	Satellites Used: Range 0 to 12
HDOP	numeric	1.01	HDOP: Horizontal Dilution of Precision
msl	numeric	499.6	MSL Altitude (m)
m	character	M	Units: Meters (fixed field)
Altref	blank	48.0	Geoid Separation (m)
m	blank	M	Units: Meters (fixed field)
DiffAge	numeric		Age of Differential Corrections (sec): Blank (Null) fields when DGPS is not used
DiffStation	numeric	0	Diff. Reference Station ID
cs	hexadecimal	*5B	Checksum
<CR> <LF>			End of message

Fix Status	Description
0	No fix / Invalid
1	Standard GPS (2D/3D)
2	Differential GPS
6	Estimated (DR) Fix

12 Appendix C. Sample Packet-Parser Code

12.1 Overview

This appendix includes sample code written in ANSI C for parsing packets from data sent by the 440 Series Inertial Systems. This code can be used by a user application reading data directly from the 440 Series product, or perhaps from a log file.

The sample code contains the actual parser, but also several support functions for CRC calculation and circular queue access.:

- **process_xbow_packet** – for parsing out packets from a queue. Returns these fields in structure XBOW_PACKET (see below). Checks for CRC errors
- **calcCRC** – for calculating CRC on packets.
- **Initialize** - initialize the queue
- **AddQueue** - add item in front of queue
- **DeleteQueue** - return an item from the queue
- **peekWord** - for retrieving 2-bytes from the queue, without popping
- **peekByte** – for retrieving a byte from the queue without popping
- **Pop** - discard item(s) from queue
- **Size** – returns number of items in queue
- **Empty** – return 1 if queue is empty, 0 if not
- **Full** - return 1 if full, 0 if not full

The parser will parse the queue looking for packets. Once a packet is found and the CRC checks out, the packet's fields are placed in the XBOW_PACKET structure. The parser will then return to the caller. When no packets are found the parser will simply return to the caller with return value 0.

The XBOW_PACKET stucture is defined as follows:

```
typedef struct xbow_packet
{
    unsigned short packet_type;
    char           length;
    unsigned short crc;
    char           data[256];
} XBOW_PACKET;
```

Typically, the parser would be called within a loop in a separate process, or in some time triggered environment, reading the queue looking for packets. A separate process might add data to this queue when it arrives. It is up to the user to ensure circular-queue integrity by using some sort of mutual exclusion mechanism withing the queue access funtions.

12.2 Code listing

```
#include <stdio.h>

/* buffer size */
#define MAXQUEUE 500

/*
 * circular queue
 */
typedef struct queue_tag
{
    int count;
    int front;
    int rear;
    char entry[MAXQUEUE];
} QUEUE_TYPE;

/*
 * crossbow packet
 */
typedef struct xbow_packet
{
    unsigned short packet_type;
    char                    length;
    unsigned short crc;
    char                    data[256];
} XBOW_PACKET;

QUEUE_TYPE circ_buf;

/*************************************************************************
 * FUNCTION:  process_xbow_packet looks for packets in a queue
 * ARGUMENTS: queue_ptr: is pointer to queue to process
 *                    result: will contain the parsed info when return value is 1
 * RETURNS:    0 when failed.
 *                    1 when successful
 *************************************************************************/
int process_xbow_packet(QUEUE_TYPE *queue_ptr, XBOW_PACKET *result)
{
    unsigned short myCRC = 0, packetCRC = 0, packet_type = 0, numToPop=0, counter=0;
    char packet[100], tempchar, dataLength;

    if(Empty(queue_ptr))
    {
        return 0;  /* empty buffer */
    }

    /* find header */
    for(numToPop=0; numToPop+1<Size(queue_ptr) ;numToPop+=1)
    {
        if(0x5555==peekWord(queue_ptr, numToPop)) break;
    }

    Pop(queue_ptr, numToPop);
```

```
    if(Size(queue_ptr) <= 0)
    {
        /* header was not found */
        return 0;
    }

  /* make sure we can read through minimum length packet */
  if(Size(queue_ptr)<7)
  {
    return 0;
  }

  /* get data length (5th byte of packet) */
  dataLength = peekByte(queue_ptr, 4);

  /* make sure we can read through entire packet */
  if(Size(queue_ptr) < 7+dataLength)
  {
    return 0;
    }


    /* check CRC */
    myCRC = calcCRC(queue_ptr, 2,dataLength+3);
    packetCRC = peekWord(queue_ptr, dataLength+5);

    if(myCRC != packetCRC)
    {
        /* bad CRC on packet - remove the bad packet from the queue and return */
        Pop(queue_ptr, dataLength+7);
        return 0;
    }

    /* fill out result of parsing in structure */
    result->packet_type = peekWord(queue_ptr, 2);
    result->length      = peekByte(queue_ptr, 4);
    result->crc         = packetCRC;
    for(counter=0; counter < result->length; counter++)
    {
        result->data[counter] = peekByte(queue_ptr, 5+counter);
    }

    Pop(queue_ptr, dataLength+7);

    return 1;
}

/*****************************************************************************
 * FUNCTION:  calcCRC calculates a 2-byte CRC on serial data using
 *            CRC-CCITT 16-bit standard maintained by the ITU
 *                    (International Telecommunications Union).
 * ARGUMENTS: queue_ptr is pointer to queue holding area to be CRCed
 *                    startIndex is offset into buffer where to begin CRC calculation
 *                    num is offset into buffer where to stop CRC calculation
 * RETURNS:    2-byte CRC
```

```
 ****************************************************************************/
unsigned short calcCRC(QUEUE_TYPE *queue_ptr, unsigned int startIndex, unsigned int num) {
    unsigned int i=0, j=0;
    unsigned short crc=0x1D0F; //non-augmented inital value equivalent to augmented initial value 0xFFFF

    for (i=0; i<num; i+=1) {
        crc ^= peekByte(queue_ptr, startIndex+i) << 8;

        for(j=0;j<8;j+=1) {
            if(crc & 0x8000) crc = (crc << 1) ^ 0x1021;
            else crc = crc << 1;
        }
    }
    return crc;
}


/****************************************************************************
 * FUNCTION:  Initialize - initialize the queue
 * ARGUMENTS: queue_ptr is pointer to the queue
 ****************************************************************************/
void Initialize(QUEUE_TYPE *queue_ptr)
{
    queue_ptr->count =  0;
    queue_ptr->front =  0;
    queue_ptr->rear  = -1;
}

/****************************************************************************
 * FUNCTION:  AddQueue - add item in front of queue
 * ARGUMENTS: item holds item to be added to queue
 *                    queue_ptr is pointer to the queue
 * RETURNS:    returns 0 if queue is full. 1 if successful
 ****************************************************************************/
int AddQueue(char item, QUEUE_TYPE *queue_ptr)
{
    int retval = 0;
    if(queue_ptr->count >= MAXQUEUE)
    {
        retval = 0; /* queue is full */
    }
    else
    {
        queue_ptr->count++;
        queue_ptr->rear = (queue_ptr->rear + 1) % MAXQUEUE;
        queue_ptr->entry[queue_ptr->rear] = item;
        retval = 1;
    }
    return retval;
}

/****************************************************************************
 * FUNCTION:  DeleteQeue - return an item from the queue
 * ARGUMENTS: item will hold item popped from queue
 *                    queue_ptr is pointer to the queue
 * RETURNS:    returns 0 if queue is empty. 1 if successful
```

```
 ****************************************************************************/
int DeleteQueue(char *item, QUEUE_TYPE *queue_ptr)
{
    int retval = 0;
    if(queue_ptr->count <= 0)
    {
        retval = 0; /* queue is empty */
    }
    else
    {
        queue_ptr -> count--;
        *item = queue_ptr->entry[queue_ptr->front];
        queue_ptr->front = (queue_ptr->front+1) % MAXQUEUE;
        retval=1;
    }
    return retval;
}

/****************************************************************************
 * FUNCTION:  peekByte returns 1 byte from buffer without popping
 * ARGUMENTS: queue_ptr is pointer to the queue to return byte from
 *                    index is offset into buffer to which byte to return
 * RETURNS:    1 byte
 * REMARKS:    does not do boundary checking. please do this first
 ****************************************************************************/
char peekByte(QUEUE_TYPE *queue_ptr, unsigned int index) {
    char byte;
    int firstIndex;

    firstIndex = (queue_ptr->front + index) % MAXQUEUE;

    byte = queue_ptr->entry[firstIndex];
    return byte;
}


/****************************************************************************
 * FUNCTION:  peekWord returns 2-byte word from buffer without popping
 * ARGUMENTS: queue_ptr is pointer to the queue to return word from
 *                    index is offset into buffer to which word to return
 * RETURNS:    2-byte word
 * REMARKS:    does not do boundary checking. please do this first
 ****************************************************************************/
unsigned short peekWord(QUEUE_TYPE *queue_ptr, unsigned int index) {
    unsigned short word, firstIndex, secondIndex;

    firstIndex = (queue_ptr->front + index) % MAXQUEUE;
    secondIndex = (queue_ptr->front + index + 1) % MAXQUEUE;

    word = (queue_ptr->entry[firstIndex] << 8) & 0xFF00;
    word |= (0x00FF & queue_ptr->entry[secondIndex]);
    return word;
}

/****************************************************************************
 * FUNCTION:  Pop - discard item(s) from queue
```

```
 * ARGUMENTS: queue_ptr is pointer to the queue
 *                     numToPop is number of items to discard
 * RETURNS:    return the number of items discarded
 ****************************************************************************/
int Pop(QUEUE_TYPE *queue_ptr, int numToPop)
{
    int i=0;
    char tempchar;
    for(i=0; i<numToPop; i++)
    {
        if(!DeleteQueue(&tempchar, queue_ptr))
        {
            break;
        }
    }
    return i;
}

/****************************************************************************
 * FUNCTION:  Size
 * ARGUMENTS: queue_ptr is pointer to the queue
 * RETURNS:    return the number of items in the queue
 ****************************************************************************/
int Size(QUEUE_TYPE *queue_ptr)
{
    return queue_ptr->count;
}

/****************************************************************************
 * FUNCTION:  Empty
 * ARGUMENTS: queue_ptr is pointer to the queue
 * RETURNS:    return 1 if empty, 0 if not
 ****************************************************************************/
int Empty(QUEUE_TYPE *queue_ptr)
{
    return queue_ptr->count <= 0;
}


/****************************************************************************
 * FUNCTION:  Full
 * ARGUMENTS: queue_ptr is pointer to the queue
 * RETURNS:    return 1 if full, 0 if not full
 ****************************************************************************/
int Full(QUEUE_TYPE *queue_ptr)
{
    return queue_ptr->count >= MAXQUEUE;
}
```