NMEA Parser Design

Monte Variakojis

monte@visualgps.net

www.visualgps.net

November 1, 2002

The NMEA 0813 standard for interfacing marine electronics devices specifies the NMEA data sentence structure as well as general definitions of approved sentences. However, the specification does not cover implementation and design. This article will hopefully clarify some of the design tasks needed to parse through NMEA sentences robustly. It will try to show techniques in parsing and data integrity checking. The article does assume that the reader has knowledge in software design and has experience in some sort of programming language. This article will reference the NMEA 0183 specification and it is recommended that the NMEA 0813 specification be available as a reference.

This article makes no assumption of the media that the NMEA data is acquired. The techniques here can be applied on received data from a higher abstraction layer. A simple example written in C++ demonstrates this parser design.

NMEA Protocol

NMEA data is sent in 8-bit ASCII where the MSB is set to zero (0). The specification also has a set of reserved characters. These characters assist in the formatting of the NMEA data string. The specification also states valid characters and gives a table of these characters ranging from HEX 20 to HEX 7E.

As stated in the NMEA 0183 specification version 3.01 the maximum number of characters shall be 82, consisting of a maximum of 79 characters between start of message "$" or "!" and terminating delimiter <CR><LF> (HEX 0D and 0A). The minimum number of fields is one (1).

Basic sentence format:

$aaccc,c--c*hh<CR><LF>

$	Start of sentence
aaccc	Address field/Command
","	Field delimiter (Hex 2C)
c--c	Data sentence block
*	Checksum delimiter (HEX 2A)
hh	Checksum field (the hexadecimal value represented in ASCII)
<CR><LF>	End of sentence (HEX OD OA)

NMEA Parser

Most protocols have a state machine tracking the protocol state and any errors that may occur during the data transfer. The NMEA parser we are discussing is based on a simple state machine. By using a state machine the computer can easily keep track of where it is within the protocol as well as recover from any errors such as timeouts and checksum errors.

The parsing example is designed such that a buffer can be sent to the parser along with the buffer length to maximize the computer processor efficiency. This will allow the computer to parse data when it is received. Since NMEA data is typically sent at 4800 baud, computers are often waiting for data. By having a state machine in place, partial or complete sets of NMEA data may be parsed. If only a partial set of data was received and sent to the parser, then when the rest of the data is received, the parser will complete any NMEA sentence that may have been incomplete.

Below, Figure 1 illustrates the data flow to the NMEA parser. Our design will assume two abstract software layers, NMEA protocol parser and specific NMEA sentence parser. It does not matter the number of data bytes received or if there is only a partial message. The NMEA parser will manage the data and extract individual NMEA sentences.

Copyright © 2002 Monte Variakojis

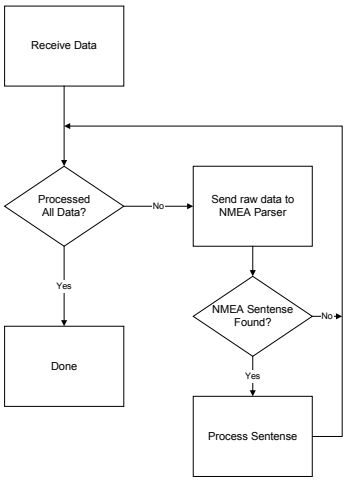

Figure 1

Each time the computer receives data it will send it to the NMEA parser. Later in this document there will be an explanation of what occurs when a valid sentence is received.

Figure 2 illustrates the NMEA protocol parser state machine. It illustrates each state and their transition. Note that there is a timeout transition in the majority of the states.

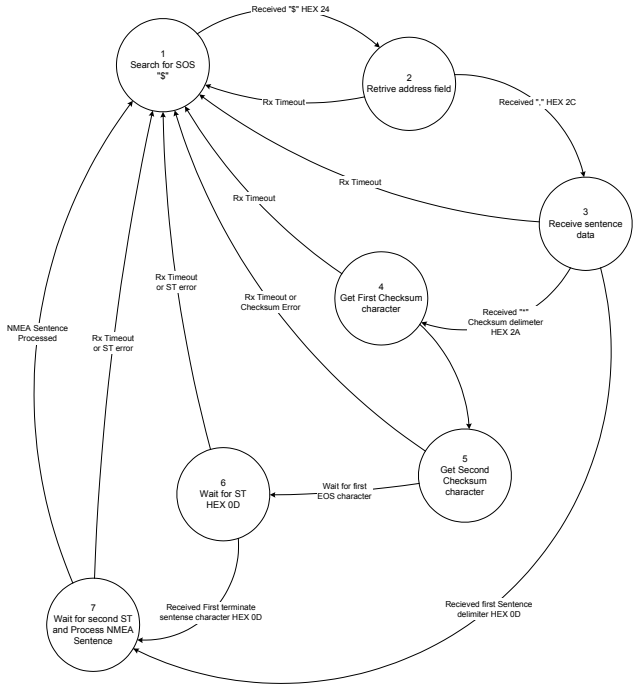

Figure 2

It may be necessary to add a timeout transition to each of the states to synchronize the state machine when no data is received for a period of time. The NMEA parser state machine illustrates this by having a transition to state 1 occur when no data has been received for a period of time. This timeout duration is application dependent. A 4800-baud NMEA device will typically send data every 1 to 2 seconds. Therefore, a timeout of 3 to 4 seconds would suffice.

State machine state definitions:

1. Search for Start of Sentence (SOS). This state will look for the '$' (HEX 24) character. When the SOS is found, then the only transition is to get the NMEA address field.
2. Retrieve address field. The parser will collect characters until it encounters a ',' (comma HEX 2C) character. We are leaving the address field flexible length. This will allow parsing of any strange or proprietary sentences.
3. Retrieve sentence data. This is where all of the data associated with the NMEA address is collected for further parsing. This state will continue to collect data and perform a checksum calculation until it receives a checksum delimiter **"*"** (HEX 2A) or sentence terminator <CR><LF> (HEX 0D 0A).
4. Receive first checksum ASCII character. This state simply waits for the first checksum character.
5. Receive second checksum ASCIII character. When the second checksum character is received, there should be enough information to verify the received checksum to the calculated checksum that was preformed in state 3. If the checksums match, then only other task to do is check for the sentence terminator.
6. Wait for first character of sentence terminator (ST). This task simply waits for the first sentence terminator character.
7. Wait for second character of sentence terminator. When the second sentence terminator is received, then the parser state machine will call the NMEA process function where the sentence address and the data are used as parameters. This function will look at the NMEA sentence address and call the appropriate sentence function.

Copyright © 2002 Monte Variakojis

Process NMEA Sentence Function

The process function is the final step in parsing the actual data. This function will simply use the NMEA sentence address to call the appropriate NMEA sentence function to extract the useful data.

Code example

There are several options when extracting data from a specific sentence. The most portable option is to extract the data, set a flag that the data has changed (typically a counter) and store it for use later. Another method could use a callback technique where the user can specify a function they would like called when the sentence is received. Our example will use the first method.

This example uses the windows™ operating system to receive NMEA data using the RS-232 serial port. The development environment is Microsoft Visual C/C++ version 6.0. The parser class is not restricted to this development environment or operating system. It has been used successfully used in embedded environments as well.

By setting up a windows timer to 100ms, the example will poll the RS-232 receive buffer and send the data to the parser. Below Figure 3 illustrates the basic flow our example will use.

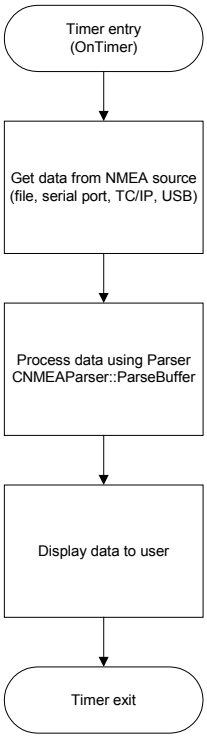

Figure 3

To simplify the windows serial port interface a wrapper class was created, CSio. This class simplifies serial port initialization, has a simple read buffer and send buffer methods.

The NMEA parser demo is a dialog-based application where the serial object (m_Sio) and NMEA parser objects (m_NMEAParser) are defined. On initialization, a 100ms timer and a 1000ms timer are initialized. The 100ms timer is used to poll the receive serial port buffer while the 1000ms timer is used to update the text in the dialog. Below is a source code segment from the timer.

```
    void CNMEAParserDemoDlg::OnTimer(UINT nIDEvent)
{
    switch(nIDEvent)
    {
        // Read data from serial port and send it to the parser
        case TIMER_ID_POLL_COMM :
            BYTE pBuff[256];
            DWORD dwBytesRead;

            //
            // Read serial port, bytes read returned
            //
            dwBytesRead = m_Sio.Read(pBuff, 255);

            //
            // Parse data returned in pBuff
            //
            m_NMEAParser.ParseBuffer(pBuff, dwBytesRead);
        break;

        // Update text in dialog
        case TIMER_ID_UPDATE_TEXT :
            UpdateText();
        break;
    }

    CDialog::OnTimer(nIDEvent);
}
```

Every 100ms (TIMER_ID_POLLCOMM) the m_Sio.Read(pBuff, 255) will read up to 255 bytes of data from the serial port buffer and place the contents into the pBuff array. The number of bytes read from the serial port buffer is returned by m_Sio.Read(). Next the m_NMEAParser.ParseBuffer() method is called to parse through the buffer and extract useful data. Every 1000ms the UpdateText() method is called to update the text on the dialog showing specific data from the NMEA parser class.

The complete NMEA parser source code is located in NMEAParser.cpp and NMEAParser.h. This class definition reflects the general parser state machine that was discussed earlier in this document. Specific NMEA sentence data is stored in individual class members defined publicly. See the NMEAParser.h file for member data definitions. In addition, each sentence that is parsed successfully, its corresponding counter is incremented. I.E. If the GGA message is received and parsed, then the GGA counter m_dwGGACount will be incremented. This is useful for determining when a specific command was received.

Source code may be found at:

http://www.visualgps.net/papers/NMEAParser.

Copyright © 2002 Monte Variakojis

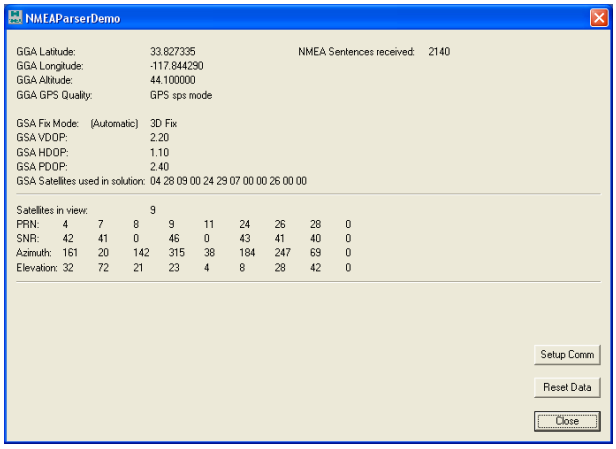

Figure 4

Figure 4 shows a screen shot of the NMEA Parser Demo windows application. Only some of the GGS, GSV and GSA data are displayed. In the lower right hand corner are controls to reset all member data to zero and to allow control over the serial port.

References

[1] National Marine Electronics Association, “NMEA 0183 Standard For Interfacing Marine Electronic Devices,” Version 3.01, January 1, 2002

Copyright © 2002 Monte Variakojis