TSIP and NMEA Application Developer's Guide

Trimble

Revision A
May 2005

Corporate Office

Trimble Navigation Limited
645 North Mary Avenue
Post Office Box 3642
Sunnyvale, CA 94088-3642
U.S.A.
Phone: +1.408.481.7920
Toll-free: +1.800.865.4851
www.trimble.com

Support Offices

For support in Europe, call:
+44.1256.746.221
or send a fax to:
+44.1256.760.148
For support outside Europe, call:
+1.408.481.8786
or send a fax to:
+1.408.481.2011

Release Notice

The following limited warranties give you specific legal rights. You may have others, which vary from state/jurisdiction to state/jurisdiction. The following limited warranties give you specific legal rights. You may have others, which vary from state/jurisdiction to state/jurisdiction.

Software and Firmware License, Limited Warranty

This Trimble software and/or firmware product (the “Software”) is licensed and not sold. Its use is governed by the provisions of the applicable End User License Agreement (“EULA”), if any, included with the Software. In the absence of a separate EULA included with the Software providing different limited warranty terms, exclusions and limitations, the following terms and conditions shall apply. Trimble warrants that this Trimble Software product will substantially conform to Trimble’s applicable published specifications for the Software for a period of ninety (90) days, starting from the date of delivery.

Warranty Remedies

Trimble's sole liability and your exclusive remedy under the warranties set forth above shall be, at Trimble’s option, to repair or replace any Product or Software that fails to conform to such warranty ("Nonconforming Product") or refund the purchase price paid by you for any such Nonconforming Product, upon your return of any Nonconforming Product to Trimble in accordance with Trimble’s standard return material authorization procedures.

Warranty Exclusions and Disclaimer

These warranties shall be applied only in the event and to the extent that: (i) the Products and Software are properly and correctly installed, configured, interfaced, maintained, stored, and operated in accordance with Trimble's relevant operator's manual and specifications, and; (ii) the Products and Software are not modified or misused. The preceding warranties shall not apply to, and Trimble shall not be responsible for defects or performance problems resulting from (i) the combination or utilization of the Product or Software with products, information, data, systems or devices not made, supplied or specified by Trimble; (ii) the operation of the Product or Software under any specification other than, or in addition to, Trimble's standard specifications for its products; (iii) the unauthorized modification or use of the Product or Software; (iv) damage caused by accident, lightning or other electrical discharge, fresh or salt water immersion or spray; or (v) normal wear and tear on consumable parts (e.g., batteries).

THE WARRANTIES ABOVE STATE TRIMBLE'S ENTIRE LIABILITY, AND YOUR EXCLUSIVE REMEDIES, RELATING TO PERFORMANCE OF THE PRODUCTS AND SOFTWARE. EXCEPT AS OTHERWISE EXPRESSLY PROVIDED HEREIN, THE PRODUCTS, SOFTWARE, AND ACCOMPANYING DOCUMENTATION AND MATERIALS ARE PROVIDED “AS-IS” AND WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND BY EITHER TRIMBLE NAVIGATION LIMITED OR ANYONE WHO HAS BEEN INVOLVED IN ITS CREATION, PRODUCTION, INSTALLATION, OR DISTRIBUTION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT. THE STATED EXPRESS WARRANTIES ARE IN LIEU OF ALL OBLIGATIONS OR LIABILITIES ON THE PART OF TRIMBLE ARISING OUT OF, OR IN CONNECTION WITH, ANY PRODUCTS OR SOFTWARE. SOME STATES AND JURISDICTIONS DO NOT ALLOW LIMITATIONS ON DURATION OR THE EXCLUSION OF AN IMPLIED WARRANTY, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

TRIMBLE NAVIGATION LIMITED IS NOT RESPONSIBLE FOR THE OPERATION OR FAILURE OF OPERATION OF GPS SATELLITES OR THE AVAILABILITY OF GPS SATELLITE SIGNALS.

Limitation of Liability

TRIMBLE’S ENTIRE LIABILITY UNDER ANY PROVISION HEREIN SHALL BE LIMITED TO THE GREATER OF THE AMOUNT PAID BY YOU FOR THE PRODUCT OR SOFTWARE LICENSE OR U.S.$25.00. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL TRIMBLE OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES WHATSOEVER UNDER ANY CIRCUMSTANCE OR LEGAL THEORY RELATING IN ANY WAY TO THE PRODUCTS, SOFTWARE AND ACCOMPANYING DOCUMENTATION AND MATERIALS, (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS), REGARDLESS WHETHER TRIMBLE HAS BEEN ADVISED OF THE POSSIBILITY OF ANY SUCH LOSS AND REGARDLESS OF THE COURSE OF DEALING WHICH DEVELOPS OR HAS DEVELOPED BETWEEN YOU AND TRIMBLE. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Table of Contents

1 INTRODUCTION 6

2 SYSTEM REQUIREMENTS 6

3 APPLICABLE DOCUMENTS 6

4 DEVELOPER'S KIT DIRECTORY STRUCTURE 6

5 DEVELOPING MULTI-PROTOCOL APPLICATIONS 7

5.1 Overview of the Sample Source Code 7

6 DEVELOPING TSIP APPLICATIONS 7

6.1 Introduction to the Sample Source Code 8

6.2 Receiving Packets 8

6.3 Parsing Packets and Extracting Data Values 8

6.4 Processing Packet Data Values 9

7 DEVELOPING NMEA APPLICATIONS 9

7.1 Introduction to the Sample Source Code 9

7.2 Receiving Packets 9

7.3 Parsing Packets and Extracting Data Values 9

7.4 Processing Packet Data Values 10

7.5 Formatting and Sending NMEA Packets 10

8 DEMO APPLICATION FOR WINDOWS 11

8.1 Introduction 11

8.2 Rebuilding the Demo Application 11

8.3 Using the Demo Application 12

Release Information

The following changes have been made to this document.

Date	Revision	Change
May 2005	A	First release.
January 2007	B	Added NMEA support

1 Introduction

This developer's guide provides information for a software engineer to implement an application to interface to a Trimble GPS receiver communicating via the Trimble Standard Interface Protocol (TSIP) and/or the National Marine Electronics Association (NMEA) protocol.

The document provides detailed descriptions of the included source code that can be adapted to the user application. In addition to this document, all the necessary components are also described and documented throughout the provided source code.

The sample source code can be compiled on a Windows PC into a demo application to interface to the GPS receiver via a serial COM port on the PC.

2 System Requirements

- The source code can be perused on any platform using a text editor.
- Microsoft Visual C++ v6.0 or .NET 2003 v7.1 is required to compile the sample source code into an executable program (demo application).
- The demo application provided as part of this developer's kit or compiled from the provided source code can run on Microsoft Windows XP, 2000, or 98.

3 Applicable Documents

[R-1] TSIP Protocol Reference (included as part of the product's user's manual)
[R-2] NMEA Protocol Reference (relevant portions included as part of the product's user's manual. The full reference is available for purchase from nmea.org)

4 Developer's Kit Directory Structure

The developer's kit has the following folder/directory structure:

```
.\---+
     |
     +--- Documents
     |
     +--- Executables
     |
     +--- Source Code
```

- `Documents` contains this guide and other relevant documentation (if applicable).
- `Executables` contains *TsipDemo.exe* which is a Windows GUI program built from the sample source code using Microsoft Visual C++. Refer to Section 8 for details on how to configure and run this program.
- `Source Code` contains various source files that show how to develop a TSIP and/or NMEA application. Section 5 provides more details about files in this directory.

5 Developing Multi-Protocol Applications

The demo application is implemented as a multi-protocol application. This means that it can monitor both TSIP and NMEA simultaneously. This is possible when the supported protocols use separate packet delimiter characters, generally don't contain each other's delimiters as part of the payload and there are generally few communication errors.

Supporting multiple protocols at the same time involves:

1. Receive bytes from the communication interface, checking each if it is the start of a packet for one of the supported protocols.
2. Once the start of a packet and the associated protocol have been identified, feed bytes into the parser for that protocol until it has received the full message.
3. Extract the data values from the received packet.
4. Pass the data values to the application for processing (e.g. to display on a screen, log to a file, etc)

Repeat the above steps for each received packet.

5.1 Overview of the Sample Source Code

The parsing-portion of the sample source code is arranged into three separate classes:

- CMetaParser. Handles the support for multiple protocols, delegating the actual work to the specific protocol parser classes.
- CTsipParser. A utility-class that handles the TSIP-specific packet parsing.
- CNmeaParser. A utility-class that handles the NMEA-specific packet parsing.

The MetaParser handles the reading of bytes from the serial port and feeds the read characters into its TsipParser and NmeaParser as appropriate (refer to ReceiveAndParsePkt() in MetaParser.cpp). An object of this class is created and used by the Demo application to display various TSIP and NMEA data in a small window on the PC screen.

The above steps are performed in the file *TsipWindow.cpp* in the function *MonitorPort()*. This function runs continuously in a thread and uses the *CMetaParser* object to complete Steps 1-3. After that, it performs Step 4 by passing the output values (formatted as an ASCII string) to the main window to be displayed on the screen. Section 8 provides more details about this application.

If you only plan to support one protocol in your application, the functionality provided by the MetaParser can be moved to the code handling the chosen protocol.

6 Developing TSIP Applications

If you only plan on supporting TSIP, the steps reduce to:

1. Receive and retrieve a TSIP packet from the communication interface. The interface is typically a serial communication port (UART).

2. Extract data values from the TSIP packet.
3. Pass the data values to the application for processing (e.g. to display on a screen, log to a file, etc).

Repeat the above steps for each received TSIP packet.

6.1 Introduction to the Sample Source Code

The source code that showcases receiving and processing TSIP packets is located in the `Source Code` directory in the file *TsipParser.cpp*. The TSIP processing functionality is implemented as a C++ class *CTsipParser* in this file. However, the individual class methods can be easily converted to regular C routines if required. The *CTsipParser* class contains routines necessary to receive a TSIP packet, parse it, and extract individual data values. An object of this class is created and used by the CMetaParser object to parse TSIP data.

6.2 Receiving Packets

Refer to the functions *ReceiveByte()* in *TsipParser.cpp and ReceiveAndParsePkt() in MetaParser.cpp*. They shows how to retrieve a TSIP packet by reading bytes from a serial port. The serial port in this example is a pointer to a C++ class encapsulating functionality of a Windows serial port. In the case of an embedded C application, however, this should be replaced by a reference to the used serial driver.

Note: one of the assumptions in this code is that the serial driver function *Read()* actually blocks the calling task/thread until a data byte is received on the port. The *ReceiveAndParsePkt()* routine is written with that assumption in mind. If the *Read()* function of your particular serial port driver is non-blocking and returns immediately regardless of whether a valid data byte was retrieved from the port or not, *ReceiveAndParsePkt()* needs to be modified to take that behavior into account.

Receiving a valid TSIP packet is done in *ReceiveByte()* as a state machine. The state machine approach is used to properly separate consecutive TSIP packets and to take care of stuffed bytes. Refer to the TSIP reference manual for more information on the TSIP protocol packet structure.

6.3 Parsing Packets and Extracting Data Values

In the previous step, the function *ReceiveByte()* filled a specified memory buffer with a valid TSIP packet and returned the TSIP packet size. This TSIP packet now needs to be parsed, which is handled by the function *ParsePkt()* in *TsipParser.cpp*. This function first extracts the packet ID from the packet buffer and then passes control to specific TSIP parsers based on the packet ID. The sample source code in *TsipParser.cpp* provides examples of parsing most TSIP packets that can be automatically output by any Trimble GPS receiver.

Each individual parser function (e.g. *Parse0x41()* for TSIP 0x41 processing) performs the following actions.

1. The parser checks that the number of data bytes in the packet matches the known number of data bytes as indicated in the TSIP reference manual.

2. Next, the parser extracts values of different data types from the packet buffer. The packet buffer is passed in using a pointer to a byte string. However, the buffer may contain values of different types of data such as a single precision floating point number or a 32-bit integer. To extract these values from the buffer, a set of data value extraction routines are provided in the file *TsipParser.cpp*. These routines are: *GetShort(), GetUShort(), GetLong(), GetULong(), GetSingle(),* and *GetDouble().*

3. Finally, the parser formats the extracted data values into an ASCII string for visual display in the program window for use in the TSIP Demo application. In the case of an embedded user application, this would be modified as required by the application that uses the GPS data (e.g. display on an LCD screen, log to a file, etc).

6.4 Processing Packet Data Values

In the case of the TSIP Demo application, the function *ParsePkt()* in *TsipParser.cpp* returns an ASCII-formatted string containing the extracted TSIP data values. The actual user application typically uses these values in a different fashion to suit specific application requirements.

7 Developing NMEA Applications

Developing an NMEA application is very similar to developing a TSIP application. All the basic steps are the same, but the actual protocol format differs somewhat. Please refer to section 6, Developing TSIP Applications for the outline of these steps.

7.1 Introduction to the Sample Source Code

The sample code handling NMEA is in NmeaParser.cpp. The structure of the CNmeaParser class is very similar to the CTsipParser class, but the functions have been adapted for NMEA.

7.2 Receiving Packets

The functional structure for receiving NMEA packets is identical to that for receiving TSIP packets, so please refer to that section while referring to the code in NmeaParser.cpp instead of TsipParser.cpp.

7.3 Parsing Packets and Extracting Data Values

The ParsePkt() function in NmeaParser.cpp handles the parsing of the received packets. It:

- Checks the checksum of the received packet
- Identifies which specific NMEA packet parsing function is to be invoked
- Returns an ASCII-formatted string containing the result.

The individual NMEA packet parsing functions have very similar structure. They read off the fields in the packet using the GetNextFieldXXX() functions and puts the results in local variables.

7.4 Processing Packet Data Values

After decoding all the data, the individual NMEA packet parsing functions pretty-print the contents to an ASCII-string that is returned to the calling function for display in the program window. The actual user application typically uses these values in a different fashion to suit specific application requirements.

7.5 Formatting and Sending NMEA Packets

The CNmeaParser class can send a sample packet to the connected GPS device with the SendPkt() function. This function is intended to demonstrate how NMEA packets can be formatted using a set of helper functions, using the Trimble proprietary VR packet as a sample.

The general outline for composing an NMEA packet is:

- Format the header. See FormatNmeaMsgHead() in NmeaParser.cpp.
- Add the relevant fields that constitute the payload of the packet. See PutNextField() in NmeaParser.cpp. (Currently there is only one such helper function implemented, but it can be extended analogously to the GetNextFieldXXX() set of functions)
- Format the tail of the packet, including the checksum. See FormatNmeaMsgTail() in NmeaParser.cpp

8 Demo Application for Windows

8.1 Introduction

The source code provided with the TSIP and NMEA Application Developer's Kit can be compiled on a Windows PC using Microsoft Visual C++ development environment. The generated executable can be used to connect to a Trimble GPS receiver, receive TSIP and NMEA packets automatically sent by the receiver, and display the packet data in a user-readable format in a window.

A pre-compiled version of the demo application is provided in the `Executables` directory. Refer to Section 8.3 for details on how to configure and run the program.

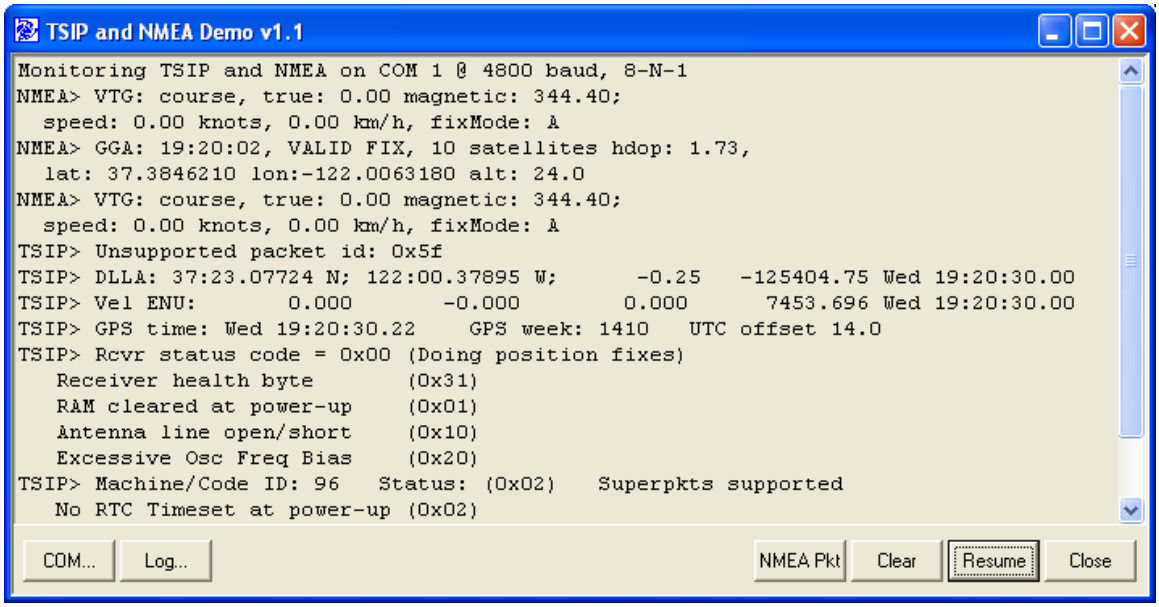

Figure 1. Demo application user interface

8.2 Rebuilding the Demo Application

- If using Microsoft Visual C++ v6.0, load the workspace file *TsipDemo.dsw* located in the `Source Code` directory. Choose *Build* | *Rebuild All* to generate *TsipDemo.exe* in the `Executables` directory.
- If using Microsoft Visual .NET 2003 v7.1, load the solution file *TsipDemo.sln* located in the `Source Code` directory. Choose *Build* | *Rebuild TsipDemo* to generate *TsipDemo.exe* in the `Executables` directory.

8.3 Using the Demo Application

Requirements

- A Trimble GPS receiver with automatic TSIP and/or NMEA output. The output port of the GPS receiver must be connected to a COM port on a Windows PC.

Configuring and Running the Program

1. Start *TsipDemo.exe* in the `Executables` directory.
2. Click on the *COM…* button, choose the COM port to which the output port of the Trimble GPS receiver is connected, and set the communication parameters as appropriate. Refer to the product manual for details on the default communication parameter settings.
3. Click on the *Start* button. You should see periodic output similar to the one shown in Figure 1 above.

Using Program Features

1. To log the received data as either binary (raw) data or as ASCII-formatted strings (as displayed in the output window), click on the *Log…* button, select appropriate checkboxes and files names and click *Start Logging*. The data will be logged in the directory where *TsipDemo.exe* is located.
2. To clear the output window, click on the *Clear* button.
3. To pause/resume the output in the window, toggle the *Pause* button.
4. To send an NMEA sample packet, click the *NMEA pkt* button.