Open Signals, Open Software: Two Years of Collaborative Analysis using the GPS Toolkit

R. Benjamin Harris, Timothy Craddock, Tracie Conn, Thomas Gaussiran, Eric Hagen, Anthony Hughes, Jon Little, Richard Mach, Scot Nelsen, Brent Renfro, Brian Tolman
Applied Research Laboratories, The University of Texas at Austin

BIOGRAPHY

R. Benjamin Harris is an Engineering Scientist at Applied Research Laboratories (ARL:UT). He received a B.S. in Aerospace Engineering from UT Austin (1994), and is a Ph.D. candidate in the same department. He received an M.S. in Aeronautics and Astronautics from Stanford (2000).

Timothy Craddock is an undergraduate majoring in the Electrical Engineering at UT Austin.

Tracie Conn is an Engineering Scientist Associate at ARL:UT. She graduated with a B.S. in Aerospace Engineering from UT Austin (2006).

Thomas Gaussiran is the director of the Space and Geophysics Laboratory (SGL) at ARL:UT. He received his B.S. in Physics from UT Austin (1988) and M.S. (1991) and PhD. (1994) from Rice University.

Eric Hagen is an undergraduate in the Aerospace Engineering department at UT Austin.

Anthony Hughes is an Engineering Scientist Associate at ARL:UT. He graduated with a B.S. in Computer Science from UT Austin (1995).

Jon Little is a Senior Engineering Scientist at ARL:UT. He obtained a B.S. (1988) and a M.S. (1990) from Auburn.

Richard Mach is a project lead at ARL:UT. He has been involved with GPS applications since 1990. He holds a B.S. (1990) and M.S. (1992) in Aerospace Engineering from UT Austin.

Scot Nelsen is an Engineering Scientist at ARL:UT. He earned a B.S. in Electrical Engineering at UT Austin (1998).

Brent Renfro is a division head at ARL:UT. He has a B.A. in Physics from Wabash College (1979) and a M.A. in Computer Science from UT Austin (1983).

Brian Tolman is a Research Scientist at ARL:UT. He holds a Ph.D. in theoretical physics from UT Austin (1982).

ABSTRACT

The GPS Toolkit, or GPSTk, is an open source project that provides a software suite to the GNSS community. The goal of the project is to free researchers to focus on research, not lower-level coding. The suite is composed of a library and a suite of applications that use that library. The library provides common routines such as those developed in satellite navigation texts. The applications support greater depth of functionality to support research and development. The applications are almost entirely console based (i.e., without a graphical user interface). They can be grouped functionally into a number of categories: basic transformations of time and coordinate systems, observation data collection and conversion, file comparison and validation, data editing, ionosphere modeling, and autonomous and relative positioning. New applications have been added to the suite over the last year. One new application, `DDBase`, can be used to generate baseline solutions with millimeter level accuracy. The library has been enhanced in the last year and low level input and output capability for BINEX data has been added. Future contributions to the library include support for the Receiver Independent Exchange Format (RINEX) Version 3 format.

INTRODUCTION

The goal of the GPSTk project is to provide a open source library and suite of applications to the satellite navigation community. GPSTk is available both as source and binaries for several platforms including Linux, Windows, and Solaris.

The GPSTk can be downloaded over the web through the project website[1]. The source is provided under the Lesser GNU Public License (LGPL). The LGPL grants the user permission to use, modify, and redistribute GPSTk source code. It does not require that works based on the GPSTk adopt an open source license. Notably, commercial projects can adopt the GPSTk without being required to open their code base.

The project source is organized into a core library and a set of applications that utilize this library. The emphasis of prior papers has been to announce the project, describe its purpose, and explain how to use the library to build new applications[2, 3]. One paper delineates how to use the applications in tandem to model the ionosphere[4]. In contrast to prior papers, this paper will contain a comprehensive introduction to the GPSTk applications for general use. In addition, recent enhancements to both the library and applications will be described. Recent changes to project operation will also be documented.

PROJECT DESCRIPTION

As a project, the GPSTk has changed significantly since it was presented at the ION-GNSS-2005. New processes have been implemented to support both development of code and use of the applications. For development, the code is now hosted in a shared online repository at Source Forge[5], and automated testing is provided by a new framework. Users are now supported with an online collaborative project site. In addition, a user manual for the GPSTk applications has been initiated and is publicly available.

Test Process

As an open source project, the source of the GPSTk is subject to intermittent updates, contributions, and corrections. The GPSTk library testing process has been redesigned to build confidence in the functionality of the library. The open source nature of the project lends need to verify that newly submitted or corrected code does not compromise existing code. Furthermore, extensive GPSTk library testing assists with verification when porting the library to new platforms or new development environments. Testing within the GPSTk library is designed with three distinct goals in mind: testing is repeatable with a low amount of effort, testing is distributed along with the library to support both internal regression testing and to assure outside users and contributors of the quality of the library, and testing is designed accommodate easy additions to the existing test suite.

GPSTk testing implements tools to both assist in building a comprehensive testing framework and to evaluate the extent to which the test suite tests the library. CppUnit [6], gcov [7], and Perl [8] scripts are tools used in the implementation of GPSTk testing. CppUnit is an open source C++ unit testing framework that is used within GPSTk. CppUnit provides the backbone for the GPSTk test suite by allowing for easy test addition and manipulation, a clear pass/fail output, and many extra features. Unit test coverage of the GPSTk library is evaluated by the GNU test coverage tool gcov. Test coverage is assessed by gcov through execution of the individual test. gcov provides an easy to read percentage of lines covered while also providing an in depth line-by-line analysis of the tested code. By examining the results of gcov, contributors are able to evaluate the thoroughness of the their unit tests and the extent of the unit tests already implemented within the GPSTk.

GPSTk testing is an ongoing process with coverage that is always expanding. Currently, GPSTk testing covers 40% of the classes within the library with an average of 95% line coverage within each of those classes. Future plans include expanding coverage over the library with the hope of eventually reaching 100% coverage of classes in the library.

Online Source Management

Prior to July 2006, access to the GPSTk source was provided via compressed snapshots. External contributions were taken via email and merged with a repository internal to ARL:UT. That process proved to be labor intensive and prone to errors. Now

Online Project Documentation

With the initial release of GPSTk, ARL:UT provided a full copy of the project's source code to the community but the development process remained inside ARL:UT. As the project has evolved, ARL:UT has been moving to an open development process. This involved moving the hosting of the code from our internal storage to a subversion repository hosted on Sourceforge. However, as with any project, there are things that need to get described and documented to help users and contributors effectively utilize or work on the GPSTk. Previously, this information was hosted on static web pages on Sourceforge or in internal documentation. To support the collaborative environment, ARL:UT is establishing a GPSTk Wiki site so that anyone can modify and contribute to this supporting material.

The initial version of the wiki is expected to contain the content of the static web pages and internal development information. This includes the GPSTk overview, instructions, examples, coding style guides, and development processes. ARL:UT has selected TWiki [9] for this project and expects to have the wiki on-line by the end of 2006. TWiki is an easy to use flexible web based collaboration platform and knowledge management system. In addition to having the ability to handle unstructured content, TWiki contains features of a structured wiki that can be used to manage structured content like defect tracking, feature requests, knowledge base, and document management. ARL:UT is working toward merging all non-code structured and unstructured GPSTk content into the wiki. This will provide an adaptive, comprehensive, and unified source of information to support the community of GPSTk developers and users.

REVIEW OF THE GPSTK APPLICATIONS

While the library provides basic functions such as those that might be found in a textbook, the GPSTk applications provide more advanced capabilities necessary to perform research and development.

The GPSTk applications are often developed to meet specific needs. Because of this, they may not compile or run under all the platforms for which the GPSTk library compiles due to platform specific dependencies.

The GPSTk applications are summarized, by group, in Table 1. They are sorted loosely into groups by their general purpose. Each row describes one GPSTk application or a set of similar applications. The application is named and described. An example of how the application is executed on a console is also provided. More detail can be found for each of these applications by referring to the GPSTk user guide. This guide describes the theory of processing behind the GPSTk applications and contains an alphabetized guide to each GPSTk application. The guide, still in development as of this writing, can be downloaded from the GPSTk website[10]. Each group is discussed briefly in the following sections.

Basic Transformations

The Basic Transformations applications are used to convert time and coordinates among various systems. The programs `timeconvert` and `calgps` can be used to convert a given time among a number of time formats: Modified Julian Date (MJD), calendar time, GPS week and seconds of week, Zcount, and others. User positions can be transformed using `poscvt`. Satellite locations can be generated in Earth-Centered, Earth-Fixed (ECEF) coordinates or in topocentric coordinates using `wheresat`. The application `wheresat` will be discussed in more detail in the "New Applications" section.

Data Collection and Conversion

The Data Collection and Conversion applications convert observations between RINEX and other formats. Some applications such as `novaRinex` and `rtAshtech` parse messages specific to a particular family of receivers. Other applications such as `RinexDump` and `navdmp` reformat RINEX into human- or machine-readable formats.

There are two sets of conversion applications that support formats derived by ARL:UT. The first set supports a format known as Floating-Integer-Character (FIC). FIC was first proposed in the 1980's as a format to record a broad range of measurements from a GPS receiver or a reference station [11], [12]. The full FIC format supports the recording of pseudorange observations and navigation messages, as well as atomic clock models. In practice, FIC is only used to store the navigation message. The second set of applications use the Measurement Data Port (MDP) protocol. The MDP format and its associated tools will be discussed in the "New Applications" section.

Comparison and Validation

The Comparison and Validation applications validate the contents of an observation or ephemeris file. Applications such as `rowdiff` difference two files of the same data. The intent of such tools is to detect failures associated with data recording software. Other applications like `rowcheck` verify the format associated with a file. Observations quality can be assessed by differencing expected observations from their computed or expected values. The application `reszilla` can perform this kind of assessment using not only RINEX based observations and ephemerides, but also using precise ephemerides.

Data Editing

The Data Editing applications merge, cut, and thin given input files. Programs such as `mergeRinObs` combine multiple files to create a new one. `ResCor` and `DiscFix` can be used to edit data associated with specific satellites or conditions.

Ionosphere Modeling

The Ionosphere Modeling applications `IonoBias` and `TECMaps` are meant to be used to together to create maps of the ionosphere. The details of using these tools to generate total electron content (TEC) maps is outside the scope of this paper but is documented elsewhere [4].

Positioning

The Positioning applications generate position estimates based on recorded observations and input ephemerides. The input ephemerides can be broadcast or precise. The applications `PRSolve` and `rinexpvt` generate autonomous positions. The application `vecsol` computes a baseline. Finally, the applications `DDBase` generates a network solution. The program `DDBase` will be discussed in more detail in the "New Applications" section.

	Tool	Description	Execution Example
Transforms	calgps	generates a GPS calendar	`calgps -Y 2004`
	poscvt	converts a given input position to other position formats	`poscvt --geodetic="30.28 262.26700 167.64"`
	timeconvert	converts given input time to other time formats	`timeconvert --calendar="07 04 2006"`
	wheresat	outputs expected location of a satellite	`wheresat -b arl2100.06n -p 3`
Collecting & Converting	rtAshtech	records observations from an Ashtech receiver	`rtAshtech -p /dev/ttyS1 -o "minute%03j%02H%02m.%06yo"`
	ficfica ficafic fic2rin	convert fic files between ASCII, binary, and RINEX formats	`fic2rin fic2100.06 rin121.06n`
	mdp2fic mdp2rinex	convert MDP files to FIC or RINEX files	`mdp2rinex -i mdpfile -o arl2100.06o`
	novaRinex	convert Novatel files to RINEX	`novaRinex --input nova2100.06 --obstype L1`
	navdmp	dumps information from nav files to human readable formats	`navdmp -i arl2100.06n -o arl2100.06.dmp`
	RinexDump	dumps observation data for specified satellites from a RINEX file	`RinexDump arl2100.06o 3 4 L1 L2`
Comparing & Validating	ephdiff	compares the satellite positions from two ephemeris sources	`ephdiff arl2100.06n fic2100.06`
	ficdiff	compares contents of two FIC files	`ficidff fic12100.06 fic22100.06`
	ficcheck ficacheck	reads a FIC file and checks it for errors reporting the first found	`ficcheck fic2100.06 -t "07/20/2006 11:00:00"`
	rowdiff rnwdiff rmwdiff	compares contents of two RINEX files	`rowdiff arl1210.06o arl22100.06o`
	rowcheck rnwcheck rmwcheck	read Rinex files and checks it for errors reporting the first found	`rnwcheck arl210.06n -e "07/20/2006 11:00:00"`
	navsum RinSum	summarizes the contents of nav/Rinex files	`RinSum -i arl2100.06o --EpochBeg 2006,07,20,13,20,00`
	mdptool	summarizes MDP data	`mdptool -i mdpfile --pvt --obs`
	reszilla	computes range residuals or zero baseline differences	`reszilla -o arl210.06o -e arl2100.06n`
Editing Data	mergeFIC	sorts and merges input FIC files into a single file	`mergeFIC -i fic12100.06 -i fic22100.06 -o ficmerge2100.06`
	mergeRinObs, -Nav, -Met	sorts and merges RINEX files	`mergeRinNav -i arl2100.06n -i arl2110.06n arl210-211.06n`
	NavMerge	merges RINEX nav files into a single file	`NavMerge -oarlnavs.06n arl2100.06n arl2110.06n`
	rinexthin	decimates an input RINEX observation files to desired data rate	`rinexthin -f arl2100.06o -s 30 -o arl2100thin.06n`
	ResCor	edits RINEX files and computes corrections	`ResCor -IFarl2100.06o -OFarl2100mod.06o -DS12,12:00:00`
	DiscFix	cycle slip corrector	`DiscFix -i arl2100.06o --DT 1.5`
Iono	IonoBias	solves interfrequency biases and a simple ionosphere model	`IonoBias --input arl2100.06o --nav arl2100.06n --XSat 3`
	TECMaps	creates maps of Total Electron Content (TEC)	`TECMaps --input arl2100.06o --nav arl2100.06n --LinearFit`
Positioning	PRSolve	generates autonomous position solution	`PRSolve -o arl2100.06o -n arl2100.06nn --XPRN 12`
	rinexpvt	generates autonomous position solution	`rinexpvt -o alr2100.06o -n arl2100.06n`
	DDBase	computes a network solution using carrier phase	`DDBase ...  --ObsFile arl2100.06o --PosXYZ x,y,z,1 --Fix`
	vecsol	estimates short baseline using range or carrier phase	`vecsol station12100.06o station22100.06o`

Table 1 GPSTk Applications, categorized, with execution examples.

NEW APPLICATIONS

wheresat

Given only a navigation file and specified PRN number, `wheresat` will generate the satellite position and clock correction beginning at the initial data timestamp given in the file and incrementing by fifteen minutes to the final timestamp in the file. The user may specify the antenna position in Earth-centered, Earth fixed (ECEF) XYZ coordinates, allowing `wheresat` to calculate the satellite's azimuth, elevation, and distance from the user. The time increment for processing, start, and end times may also be specified. For users intending to utilize MATLAB[13] or another program for analysis of the generated data, `wheresat` can produce a file with the resulting data in a matrix format without header information. The example command line execution shown in Figure 1 will produce the position, clock correction, azimuth, elevation, and range from the user of PRN 2 in 10 minute increments between 6:00 and 7:30 on July 19, 2006.

Several options are available that allow the user to customize the output. The full set of argument definitions is produced by running `wheresat -h` on the command line.

```
wheresat -b sampleNavFile.06n -p 2 -t 600
  -u "-740399.800 -5458071.700 3207345.600"
  -s "07/19/2006 06:00:00" -e "07/19/2006 7:30:00"
```

Fig. 1 Invoking `wheresat`

MDP Tools

The GPSTK source tree `apps/MDPtools` directory contains support for the MDP data format. It is a data format and protocol for communicating GPS data over a TCP/IP connection. It is a binary format that supports raw observations, PVT solutions, navigation data, and a simple status message. In contrast to other data formats or protocols, MDP supports not only information associated with surveying or navigation but also information to assist in satellite monitoring. This includes providing observations from all code/carrier combinations along with information regarding the tracking process itself. It is also a means of transferring all navigation bits that were demodulated from the signals; not just the elements typically used by receivers.

The MDP format is message oriented with each message starting with a fixed length header followed by a variable length body. Each header includes a frame word, message id, message length, time, and a 16 bit CRC. See the class `MDPHeader` for the details of this part of the format.

The body of the observation message is defined in the `MDPObsEpoch` class. A single message contains all the observation data from a single SV for a single epoch. Each code/carrier has a pseudorange, phase, Doppler, SNR, tracking loop bandwidth, and a lock count. IEEE double precision floating point format is used to represent the pseudorange, phase, and Doppler values. The body of the navigation data message is defined in `MDPNavSubframe` class. It stores a single subframe as broadcast from a single code/carrier from a single SV. All bits are included starting with the first bit of the TLM/preamble to the last parity/CRC bit. The body of the PVT solution message is defined in the `MDPPVTSolution` class. It includes an ECEF position and velocity. It also includes a receiver time solution and clock drift rate. The class `TCPStream` provides support for receiving MDP from a TCP socket.

Several applications are provided to work with MDP data: `mdptool`, `mdp2rinex`, `mdp2fic`, and `mdpscreen`. All programs can receive the MDP data from either a file or a TCP socket. `mpdtool` produces various summaries and tables of the provided data. `mdp2rinex` produces RINEX observation and navigation files from the data stream. `mdp2fic` produces a FIC file of all the unique navigation subframes. `mdpscreen` produces a text display based on the open source `ncurses`[14] library that is similar to the front panel display on some receivers. This display is intended to be used with a MDP stream that is being generated in real-time. Figure 2 shows `mdpscreen` running on a receiver tracking C/A- and Y-codes on L1 and Y code on L2.

DDBase

`DDBase` is a double differenced GPS carrier phase network processor designed to estimate precise relative positions at all baseline lengths. It is the product of a rapid development at ARL:UT during early 2005 in support of a project that requires determination of many short baselines with sub-millimeter precision, short datasets, little or no manual intervention, and high throughput. `DDBase` is now deployed in the field for that initial project[15]. It has been tested extensively on baselines up to 5 kilometers, both at ARL:UT and in the field, and consistently produces sub-millimeter to few-millimeter results.

`DDBase` was developed in C++ completely from scratch using the GPSTk library. It also makes extensive use of the standard C++ library. It is a stand-alone console application that runs in batch mode. It is very modular, and even includes place-holder modules for some specialized features that have not yet been implemented. It takes input from the command line and/or a configuration file and produces output to a log file and several optional intermediate data files. Input GPS measurement data is in RINEX observation files (pseudorange and phase; L1, L2 or both) and either RINEX or SP3 format navigation files. `DDBase` also reads Earth orientation parameters from a file (either NGA or IGS format) and includes an implementation of conventional terrestrial and inertial frames as described by the IERS [16, Chapter 5].

```
arl-newrx102                    24 x 80           13:24:34  8/21/06  GPS

PVT: 13:24:36.0   Offset:      86.8 ns  Drift:     56.59 ns/d
Lon: 97.72569 W   Lat: 30.38359 N   Ht: 220.994 m    Rate: 1.5 s
 Vx: 0.18 cm/s     Vy: -0.07 cm/s   Vz: -0.08 cm/s   FOM:  1   0 0
 Trx: 37C    ExtFreq: Locked     StartTime: 14:37 8/11/2006
Tant: 29C   Selftest: 0           TestTime: 13:23 8/21/2006

Obs Rate: 1.5 s
                   C1    P1      C2    P2      lock
Ch Prn   Az  El    SNR   SNR     SNR   SNR     count  iodc   h
-- ---  ---  --   ----  ------  ----  ------  ------  ----  --
 1   7  208   8+  44.9  40.0 Y        43.2 Y   11393    c7   0
 2  14  305  65+  57.4  53.4 Y        53.9 Y    7053     1   0
 3   9   42  17-  49.9  45.9 Y        42.6 Y    9033   1a2   0
 4   1  253  30+  52.0  47.8 Y        47.1 Y    2893   2a9   0
 5  19  283  16+  47.7  44.7 Y        43.5 Y    3873   1dd   0
 6  25  201  15+  47.4  43.2 Y        44.1 Y     765   16d   0
 7  21  153  26-  50.9  46.8 Y        46.0 Y   10297   1c3   0
 8  18   79  40-  56.1  52.3 Y        48.4 Y   14053   123   0
 9  15  161  51-  58.1  53.6 Y        52.4 Y   11833   16d   0
10  22   18  62+  59.1  54.5 Y        54.0 Y    9953    8a   0
11   3  251  13+  46.2  41.8 Y        41.0 Y    6873   3db   0
12  30  126   7+  42.0  37.3 Y        41.2 Y     113    55   0
```

Fig. 2 mdpscreen display

`DDBase` is documented in a help page that is output by the program itself and in a complete user's guide. The user's guide includes examples and a separate command reference.

`DDBase` preprocesses the data using the receiver autonomous integrity monitor (RAIM) pseudorange solution (toolkit class `PRSolution`) to edit anomalous points and to determine the receiver positions and clock biases. Carrier phase data is double differenced using an automatic algorithm that determines a minimal unique set of baselines and optimal (based on time and elevation angle) reference satellites. The user also has the option of specifying the reference satellites in the input. Double differenced phases are edited for cycleslips and outliers using several algorithms, including some robust statistical techniques. Tropospheric models are used from the GPSTk library. The user may choose which ones are used. The estimator in `DDBase` is the measurement update of the Square Root Information Filter form of the Kalman filter. It is an iterative, linearized least squares processor that estimates receiver positions, phase biases, and optionally residual zenith tropospheric delays. With single frequency data the biases may be fixed to their integerized estimated values on the last iteration. The user also has the option of specifying *a priori* constraints on the solution when the biases are floating and when they are fixed.

`DDBase` includes a full implementation of the Square Root Information Filter (SRIF) form of the Kalman filter algorithm[17]. Class `SRI` encapsulates the information contained in a set of simultaneous equations, in square root form, that is an upper-triangular information matrix R, a state vector z, and a namelist. A namelist is a set of human-readable labels that identify the elements of the state vector and the covariance matrix. This class includes methods that implement ordinary least squares (measurement updates and inversion to get solution and covariance), as well as the manipulation of the SRI data, that includes merging, splitting and permuting elements. Class `SRIFilter` inherits from `SRI` and implements the Kalman filter in SRIF form. `SRIFilter` may be used for Kalman filtering, smoothing, and least squares applications, including weighted, linear or linearized, robust and/or sequential algorithms.

`DDBase` has routinely produced sub-millimeter accuracy on very short baselines and few-millimeter or better repeatability on baselines in the field up to 5 kilometers in length. The results shown in Figure 4 represent the output of `DDBase` using scripts with no manual editing of the input data nor the results, and with no special configuration of `DDBase` input parameters. Figure 3(a) presents the difference between the final `DDBase` solution and a precise conventional survey ($\pm$ 0.4mm each component) of one 30 meter baseline on the antenna testbed at ARL:UT. This is for 68 consecutive 1-hour sets of 1-second L1 only data. The average offset (0.37, -0.33, 0.53 mm ECEF XYZ) is approximately at the stated accuracy of the survey, and standard deviations (0.25, 0.45, 0.34 mm ECEF XYZ) are sub-millimeter. Experience has shown that processing more data will improve these results slightly. For very short baselines, acceptable results can be achieved using at least one hour of observations. Figures 3(b) and 3(c) present results from the field on a baseline of almost 5 kilometers, differenced from the average solution. This test included 21 hours of consecutive data on two days, collected with the same hardware as in the previous test on one site and combined with data from different hardware in an established reference station at the other site. Figure 3(b) presents `DDBase` results for 1-hour consecutive datasets. Figure 3(c) presents results for the same data processed in overlapping 3-hour datasets. Repeatability is 3 mm or better (each component) with one-hour datasets, and improves to 2 mm or better with three hours of data. `DDBase` has not yet been tested at long baselines.

Improvement and further development of `DDBase` is continuing. ARL:UT is actively using `DDBase` to work on the measurement of antenna phase center variations (PCV), including ways to input more detailed (such as tabular) PCV data into `DDBase`. ARL:UT also continues to test `DDBase` at longer baselines and to implement new features. We are also very interested in the identification and mitigation of multipath within `DDBase`. We look forward to interest and involvement in `DDBase` from the GPSTk community.

LIBRARY FUNCTIONS

The GPSTk library has been enhanced since the last presentation at the ION. The code base has been modified to support GNU GCC C++ (gcc) version 4. Also, the library has been modified to support native 64 bit Linux. Many contributions, including two major ones, have been added. The library can now read and write low level BINEX streams. New ways to represent and store time have been prototyped. There are many smaller contributions that would be of interest to the general community.

Low Level BINEX Support

BINEX is a binary format for the exchange of GPS/-GLONASS/SBAS data [18]. The design of BINEX is ongoing and emphasizes flexibility and extensibility. Due to the increasing relevance of BINEX, the GPSTk has been updated to support it. Like BINEX itself, support for BINEX in the GPSTk is still evolving.

BINEX support in the GPSTk facilitates the reading and writing of files containing generic BINEX records and accessing of data in those records. The support mirrors that provided for other record-based formats in the toolkit. Two new classes are provided, one to represent an individual record and one to represent a stream of records.

- `BinexData`. This derivative of `FFData` encapsulates an individual BINEX record and provides methods for safely accessing record contents. This class includes subclasses `UBNXI` and `MGFZI` to represent BINEX values of the same name.
- `BinexStream`. This derivative of `FFBinaryStream` facilitates the reading and writing of `BinexData` objects to files.

The implementation of BINEX support in the GPSTk adheres to the object-oriented style of the toolkit while leveraging existing prototype code from UNAVCO [18]. In addition, the GPSTk implementation features robust BINEX record validation, error handling, and encapsulation. Figure 4 depicts the structure of the new BINEX classes. The diagram follows the Unified Modeling Language (UML) format [19].

Using the new BINEX classes is straightforward. To read data from a BINEX file, a user must first create and open a BinexStream then repeatedly create BinexData objects filled with data from the stream. Data can be extracted from the resulting objects at any later time. This process is illustrated in Figure 5.

New Time Classes

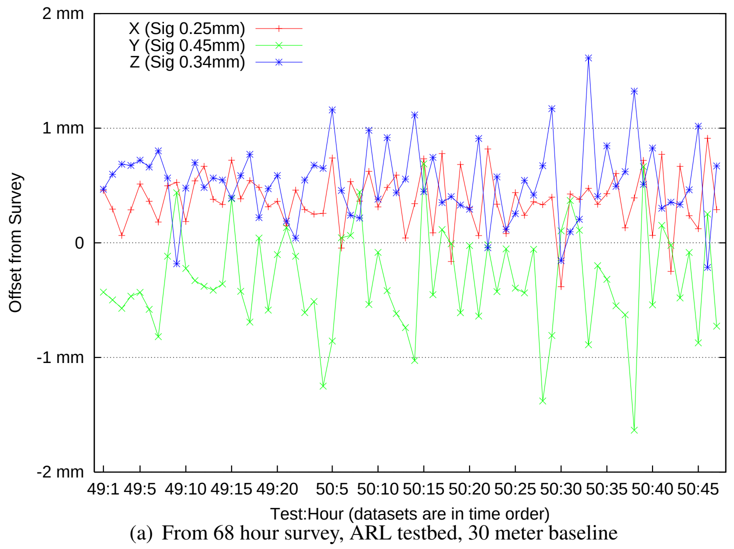
Most of the representations of time in the GPSTk library utilize the `DayTime` class. This single class is used to both convert and store time quantities. Not only is `DayTime` used to note epochs, but also to correct pseudorange and phase observations for known effects such as Earth rotation or clock drift. Therefore, the precision of `DayTime` directly limits the precision of corrected observations. Furthermore, as `DayTime` must convert most time quantities at least once, there are round-off and truncation errors associated with using time quantities in the GPSTk.

A new set of class has been prototyped that separate conversion from storage. To promote a uniform interface to all time classes, the new storage classes inherit from the class `TimeTag`. For example, the classes `GPSWeekZcount` and `MJD` derive from `TimeTag`. `GPSWeekZcount` uses two parameters to store an epoch: an `int` for the GPS week and an `int` for Z-count. The class `MJD` uses only one parameter called a `double`. Conversion among the new classes is accomplished by converting to and from the intermediate class `CommonTime`.

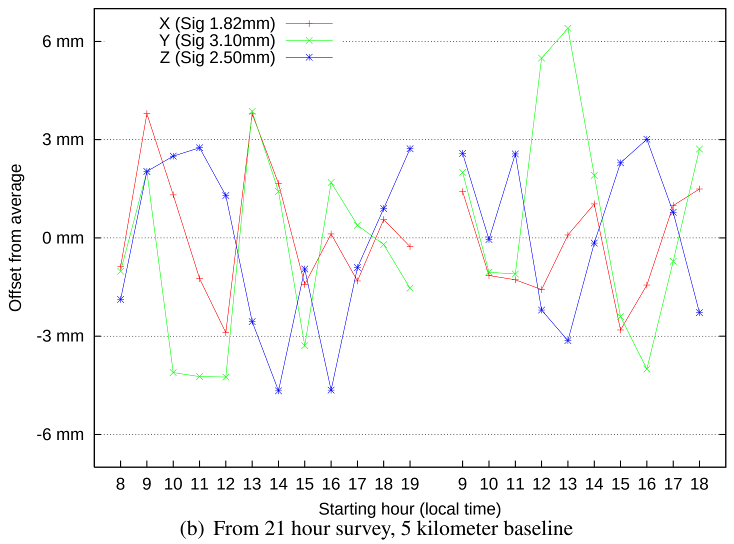
At this time these classes are provided for evaluation. When this approach has been assessed for usability and thoroughly tested, `DayTime` will be replaced throughout the library in favor of the new time classes. There will be a period of transition where `DayTime` is still provided for backwards compatibility.

Minor Changes

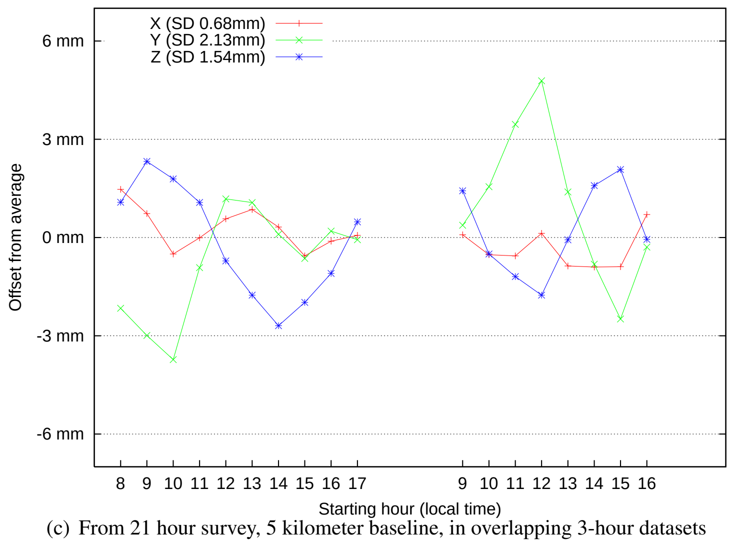
Support for the RINEX version 2.11 standard has been added [20]. In addition to other changes, RINEX 2.11 introduces the L2C pseudorange observable `C2`. Other library enhancements apply to the processing of GNSS ob-



(a) From 68 hour survey, ARL testbed, 30 meter baseline



(b) From 21 hour survey, 5 kilometer baseline



(c) From 21 hour survey, 5 kilometer baseline, in overlapping 3-hour datasets

Fig. 3 `DDBase` position solution residuals

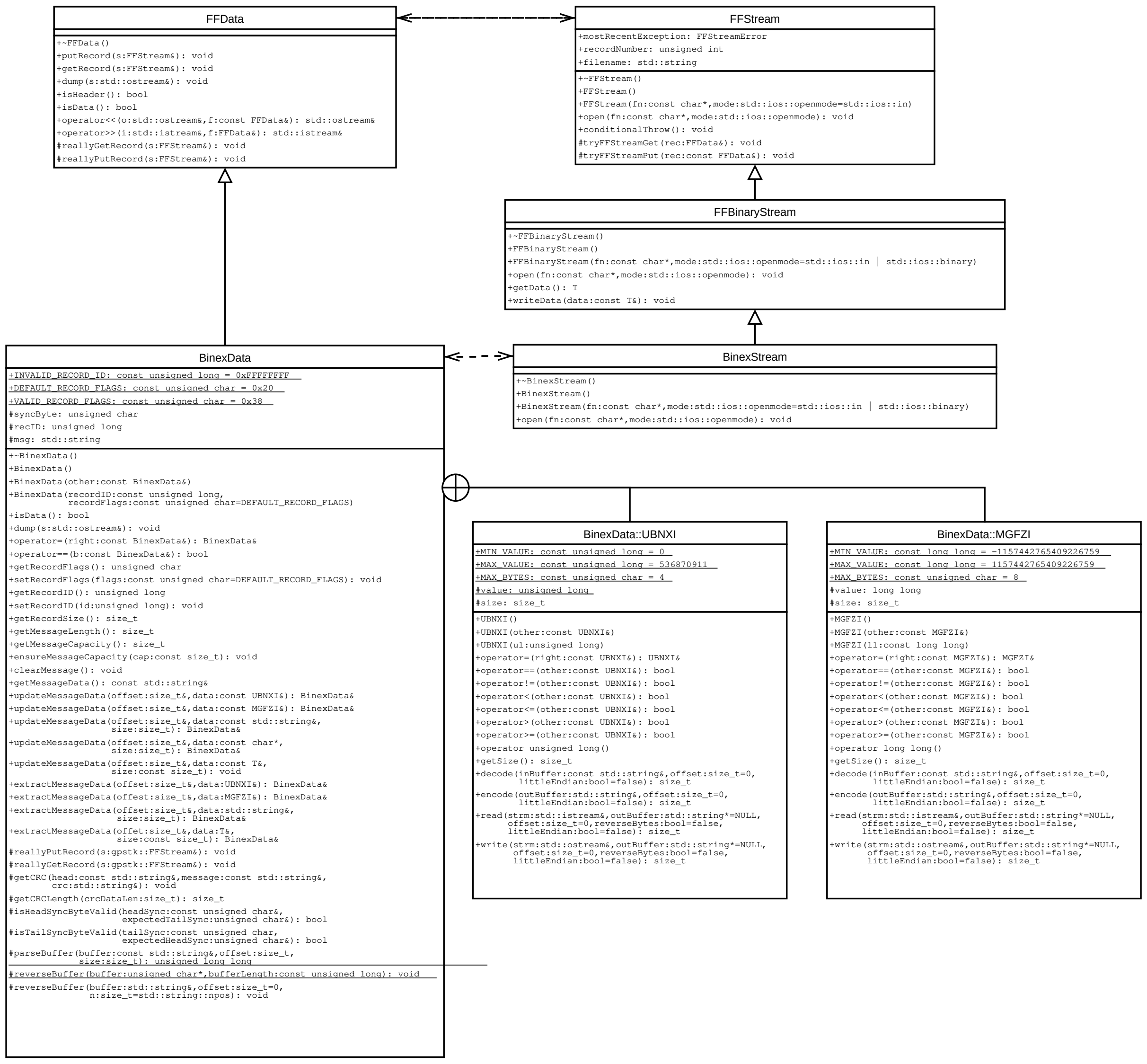

FFData
+~FFData()
+putRecord(s:FFStream&): void
+getRecord(s:FFStream&): void
+dump(s:std::ostream&): void
+isHeader(): bool
+isData(): bool
+operator<<(o:std::ostream&,f:const FFData&): std::ostream&
+operator>>(i:std::istream&,f:FFData&): std::istream&
#reallyGetRecord(s:FFStream&): void
#reallyPutRecord(s:FFStream&): void
FFStream
+mostRecentException: FFStreamError
+recordNumber: unsigned int
+filename: std::string
+~FFStream()
+FFStream()
+FFStream(fn:const char*,mode:std::ios::openmode=std::ios::in)
+open(fn:const char*,mode:std::ios::openmode): void
+conditionalThrow(): void
#tryFFStreamGet(rec:FFData&): void
#tryFFStreamPut(rec:const FFData&): void
FFBinaryStream
+~FFBinaryStream()
+FFBinaryStream()
+FFBinaryStream(fn:const char*,mode:std::ios::openmode=std::ios::in | std::ios::binary)
+open(fn:const char*,mode:std::ios::openmode): void
+getData(): T
+writeData(data:const T&): void
BinexStream
+~BinexStream()
+BinexStream()
+BinexStream(fn:const char*,mode:std::ios::openmode=std::ios::in | std::ios::binary)
+open(fn:const char*,mode:std::ios::openmode): void
BinexData
+INVALID_RECORD_ID: const unsigned long = 0xFFFFFFFF
+DEFAULT_RECORD_FLAGS: const unsigned char = 0x20
+VALID_RECORD_FLAGS: const unsigned char = 0x38
#syncByte: unsigned char
#recID: unsigned long
#msg: std::string
+~BinexData()
+BinexData()
+BinexData(other:const BinexData&)
+BinexData(recordID:const unsigned long, recordFlags:const unsigned char=DEFAULT_RECORD_FLAGS)
+isData(): bool
+dump(s:std::ostream&): void
+operator=(right:const BinexData&): BinexData&
+operator==(b:const BinexData&): bool
+getRecordFlags(): unsigned char
+setRecordFlags(flags:const unsigned char=DEFAULT_RECORD_FLAGS): void
+getRecordID(): unsigned long
+setRecordID(id:unsigned long): void
+getRecordSize(): size_t
+getMessageLength(): size_t
+getMessageCapacity(): size_t
+ensureMessageCapacity(cap:const size_t): void
+clearMessage(): void
+getMessageData(): const std::string&
+updateMessageData(offset:size_t&,data:const UBNXI&): BinexData&
+updateMessageData(offset:size_t&,data:const MGFZI&): BinexData&
+updateMessageData(offset:size_t&,data:const std::string&, size:size_t): BinexData&
+updateMessageData(offset:size_t&,data:const char*, size:size_t): BinexData&
+updateMessageData(offset:size_t&,data:const T&, size:const size_t): void
+extractMessageData(offset:size_t&,data:UBNXI&): BinexData&
+extractMessageData(offest:size_t&,data:MGFZI&): BinexData&
+extractMessageData(offset:size_t&,data:std::string&, size:size_t): BinexData&
+extractMessageData(offet:size_t&,data:T&, size:const size_t): BinexData&
#reallyPutRecord(s:gpstk::FFStream&): void
#reallyGetRecord(s:gpstk::FFStream&): void
#getCRC(head:const std::string&,message:const std::string&, crc:std::string&): void
#getCRCLength(crcDataLen:size_t): size_t
#isHeadSyncByteValid(headSync:const unsigned char&, expectedTailSync:unsigned char&): bool
#isTailSyncByteValid(tailSync:const unsigned char, expectedHeadSync:unsigned char&): bool
#parseBuffer(buffer:const std::string&,offset:size_t, size:size_t): unsigned long long
#reverseBuffer(buffer:unsigned char*,bufferLength:const unsigned long): void
#reverseBuffer(buffer:std::string&,offset:size_t=0, n:size_t=std::string::npos): void
BinexData::UBNXI
+MIN_VALUE: const unsigned long = 0
+MAX_VALUE: const unsigned long = 536870911
+MAX_BYTES: const unsigned char = 4
#value: unsigned long
#size: size_t
+UBNXI()
+UBNXI(other:const UBNXI&)
+UBNXI(ul:unsigned long)
+operator=(right:const UBNXI&): UBNXI&
+operator==(other:const UBNXI&): bool
+operator!=(other:const UBNXI&): bool
+operator<(other:const UBNXI&): bool
+operator<=(other:const UBNXI&): bool
+operator>(other:const UBNXI&): bool
+operator>=(other:const UBNXI&): bool
+operator unsigned long()
+getSize(): size_t
+decode(inBuffer:const std::string&,offset:size_t=0, littleEndian:bool=false): size_t
+encode(outBuffer:std::string&,offset:size_t=0, littleEndian:bool=false): size_t
+read(strm:std::istream&,outBuffer:std::string*=NULL, offset:size_t=0,reverseBytes:bool=false, littleEndian:bool=false): size_t
+write(strm:std::ostream&,outBuffer:std::string*=NULL, offset:size_t=0,reverseBytes:bool=false, littleEndian:bool=false): size_t
BinexData::MGFZI
+MIN_VALUE: const long long = -1157442765409226759
+MAX_VALUE: const long long = 1157442765409226759
+MAX_BYTES: const unsigned char = 8
#value: long long
#size: size_t
+MGFZI()
+MGFZI(other:const MGFZI&)
+MGFZI(ll:const long long)
+operator=(right:const MGFZI&): MGFZI&
+operator==(other:const MGFZI&): bool
+operator!=(other:const MGFZI&): bool
+operator<(other:const MGFZI&): bool
+operator<=(other:const MGFZI&): bool
+operator>(other:const MGFZI&): bool
+operator>=(other:const MGFZI&): bool
+operator long long()
+getSize(): size_t
+decode(inBuffer:const std::string&,offset:size_t=0, littleEndian:bool=false): size_t
+encode(outBuffer:std::string&,offset:size_t=0, littleEndian:bool=false): size_t
+read(strm:std::istream&,outBuffer:std::string*=NULL, offset:size_t=0,reverseBytes:bool=false, littleEndian:bool=false): size_t
+write(strm:std::ostream&,outBuffer:std::string*=NULL, offset:size_t=0,reverseBytes:bool=false, littleEndian:bool=false): size_t

Fig. 4 BINEX classes added to the GPSTk library

```
#include "BinexData.hpp"
#include "BinexStream.hpp"

// Read a BINEX file and output the first 3 fields of message data for each record
// (assuming the fields are a long, a short, and a float).
//
int main (int argc, char *argv[])
{
   gpstk::BinexStream inStream("inputfile", std::ios::in | std::ios::binary);
   inStream.exceptions(ios_base::failbit);
   while (inStream.good() && (EOF != inStream.peek() ) )
   {
      gpstk::BinexData record;
      try
      {
         record.getRecord(inStream);

         size_t offset = 0;
         long   field1;
         short  field2;
         float  field3;

         record.extractMessageData(offset, field1, sizeof(field1) );
         record.extractMessageData(offset, field2, sizeof(field2) );
         record.extractMessageData(offset, field3, sizeof(field3) );

         std::cout << "Data: " << field1 << ", " << field2 << ", "
                               << field3 << std::endl;
      }
      catch (gpstk::FFStreamError e)
      {
         std::cout << e << std::endl;
      }
      catch (...)
      {
         std::cout << "Unknown error reading record." << std::endl;
      }
   }
   inStream.close();
   return 0;
}
```

Fig. 5 Example of code using BINEX routines

servations in a general sense. New code has been added to convert satellite positions from ECEF coordinates to a radial/alongtrack/crosstrack (RAC) coordinate system.

FUTURE DIRECTIONS

The future capabilities of the GPSTk software suite depend wholly on contributions. Within ARL:UT, improvements to the GPSTk are underway. Some of these improvements are porting activities. Others projects have identified specific new functionality.

Multiple projects within ARL:UT wish to make the GPSTk software suite function on more operating systems using a larger variety of compilers. Efforts are underway to create a full featured port to the Mac OS X operating system. Also, the GPSTk is being ported to the Borland C++ series of compilers, providing another choice to those who wish to develop GPSTk based applications in Windows. Both Borland and Microsoft now offer free, lightweight versions of their commercial compilers [21], [22]. Support for both of those compilers is a goal for the GPSTk project.

RINEX provides a widely supported way to store GNSS data in a platform independent ASCII format that has been in use since the late 1980s[11]. RINEX Version 3 includes a major change to the format that is meant to handle the upcoming changes to GPS due to modernization efforts and the upcoming Galileo system [20]. Version 3 maintains a similar ASCII format while expanding the format to have more flexible and expandable observation identifiers. Internal ARL:UT projects have identified that supporting this format will be useful. ARL:UT intends to integrate support of the format into a future version of the GPSTk.

SUMMARY

After two years of operation, the GPSTk has amassed a number of contributions. Many contributions have enhanced the library to allow for improved processing of observations or ephemerides. Many of these contributions have taken the form of command line or console applications. Those applications are now available not only as source but also as binaries. These applications can be used in a variety of stages in the research and development process.

ACKNOWLEDGMENTS

The authors would like to thank ARL:UT for its long term support of the GPSTk project. Also, they would like to thank the many members of the GPSTk developer community that have assisted the project with patches, applications, and bug reports to the GPSTk project. In particular, Martin Vermeer and Dagoberto Salazar have made notable contributions.

REFERENCES

[1] GPSTk website. http://www.gpstk.org/.

[2] R. Benjamin Harris, Brian Tolman, Tom Gaussiran, David Munton, Jon Little, Richard Mach, Scot Nelsen, and Brent Renfro. The GPS Toolkit: Open Source GPS Software. In *Proceedings of the 16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Long Beach, California, September 2004.

[3] Brian Tolman and R. Benjamin Harris. The GPS Toolkit. *Linux Journal*, pages 72–76, September 2004.

[4] Tom Gaussiran, Brian Tolman, and Ben Harris. An Open Source Toolkit for GPS Processing, Total Electron Content Effects, Measurements and Modeling. In *Proceedings of the International Beacon Satellite Symposium*, Trieste, Italy, October 2004.

[5] SourceForge website. http://sourceforge.net/.

[6] *The CPPUnit Wiki*. http://cppunit.sourceforge.net/cppunit-wiki.

[7] *Introduction to GCOV*. http://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html.

[8] *The Perl Directory at perl.org*. http://www.perl.org/.

[9] *TWiki(tm) - an Enterprise Collaboration Platform*. http://www.twiki.org/.

[10] Tracie Conn, Tom Gaussiran, R. Benjamin Harris, Jon Little, Richard Mach, David Munton, Brent Renfro, Brian Tolman, and Martin Vermeer. *The GPS Toolkit: A User's Guide for Scientists, Engineers and Students*. "http://www.gpstk.org/guide".

[11] Werner Gürtner and Gerald M. Mader. *The RINEX Format: Current Status, Future Developments*. http://navcenter.org/ftp/GPS/REPORTS/rinex.txt, 1990.

[12] V.D. Scott and J. Clynch. A Proposed Standardized Exchange Format for Navstar GPS Geodetic Data. In *Proceedings of the Fourth International Geodetic Symposium on Satellite Systems*, Austin, Texas, 1986.

[13] *The Mathworks–MATLAB®–the Language of Technical Computing*. http://www.mathworks.com/products/matlab/.

[14] *Announcing ncurses 5.5*. http://www.gnu.org/software/ncurses/ncurses.html.

[15] J. Clark Hughes, Joel A. Banks, Aaron J. Kerkhoff, Dr. Brian W. Tolman, Jon R. Wyant, and Rex Ellison. Sub-millimeter Precision GPS Survey System at the Holloman High Speed Test Track. In *Proceedings of the 18th International Technical Meeting of the*

Satellite Division of the Institute of Navigation, Fort Worth, Texas, September 2006.

[16] Dennis D. McCarthy. *IERS Technical Note 21, IERS Conventions*. U.S. Naval Observatory, 1996.

[17] G. J. Biermann. *Factorization Methods for Discrete Sequential Estimation*. Academic Press, 1977.

[18] *BINEX: Binary Exchange Format*. http://binex.unavco.org/.

[19] *Unified Modeling Language Specification Version 2.0*. Object Management Group, http://www.omg.org/, August 2005.

[20] Werner Gürtner and Lou Estey. *RINEX: The Reciever Independant Exchange Format Version 3.00*. ftp://igscb.jpl.nasa.gov/igscb/data/format/rinex300.pdf, 2006.

[21] *Borland C++ Compiler version 5.5*. http://www.borland.com/bcppbuilder/freecompiler/.

[22] *Microsoft Visual C++ 2005 Express*. http://lab.msdn.microsoft.com/express/visualc/.