ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη λογισμικού επικοινωνίας με πρότυπο δορυφορικό δέκτη GPS: NORD (NovAtel OEM4 Receiver Daemon)

Γιώργος Π. Παπαδάκης

Εξεταστική Επιτροπή:

Κώστας Μπάλας, Αναπληρωτής Καθηγητής (επιβλέπων)

Στέλιος Μερτίκας, Καθηγητής

Κώστας Καλαϊτζάκης, Καθηγητής

Χανιά 2011

Περίληψη

Σε αυτή την εργασία αναπτύχθηκε ένα λογισμικό για τον έλεγχο και τη διαχείριση των ηλεκτρονικών καρτών GPS (Global Positioning System) της σειράς OEM4 της εταιρίας NovAtel. Η κάρτα OEM4 είναι ένας δέκτης GPS, ο οποίος ενσωματώνεται σε έναν υπολογιστή και επικοινωνεί με αυτόν μέσω θυρών ευρείας χρήσης (σειριακή θύρα, USB). Σε συνδυασμό με μια κεραία και ένα ομοαξονικό καλώδιο λειτουργούν ως ένα ολοκληρωμένο σύστημα προσδιορισμού θέσης . Ένα τέτοιο σύστημα υπάρχει στο Εργαστήριο Γεωδαισίας και Γεωπληροφορικής στο Πολυτεχνείο Κρήτης, το οποίο δεν υποστηριζόταν από κάποιο λογισμικό επικοινωνίας και ελέγχου και επομένως υπήρχε η ανάγκη για τη δημιουργία ενός τέτοιου.

Δεδομένης αυτής της έλλειψης λογισμικού διαχείρισης και ελέγχου της κάρτας OEM4, αναπτύξαμε το λογισμικό NORD (NovAtel OEM4 Receiver Daemon) προκειμένου να εξασφαλιστεί η επικοινωνία με τον υπολογιστή αλλά και η απομακρυσμένη διαχείριση της κάρτας. Το λογισμικό που αναπτύχθηκε αποτελεί απαραίτητο εργαλείο για την όποια διαχείριση του γεωδαιτικού σταθμού GPS από τον χρήστη.

Η βασική δυνατότητα που παρέχεται μέσω του λογισμικού NORD, είναι να αποθηκεύει ανά χρονικά διαστήματα επιλογής του χρήστη, αρχεία παρατήρησης και πλοήγησης GPS σε μορφοποίηση RINEX (Receiver Independent Exchange Format, έκδοσης 2.11). Με άλλα λόγια, το λογισμικό μας, σε συνδυασμό με το RINEX ‘μεταφράζει’ τα πρωτογενή δεδομένα που λαμβάνονται από την κεραία του δέκτη OEM4, ώστε να είναι αναγνώσιμα από το χρήστη.

Επιπλέον, ο χρήστης δύναται να τροποποιήσει τις λειτουργίες της κάρτας OEM4 ανάλογα με τις ανάγκες του, όπως να επανεκκινήσει την κάρτα, να ενεργοποιήσει ή να απενεργοποιήσει τις σειριακές/ USB θύρες του, ή ακόμα και να αλλάξει τις ρυθμίσεις της κεραίας.

Επιπρόσθετα, η εφαρμογή NORD παρακολουθεί την εναλλαγή της ηλεκτρικής κατάστασης του ακροδέκτη RTS (Request To Send) της σειριακής θύρας επικοινωνίας της κάρτας OEM4 από υψηλό σε χαμηλό δυναμικό και αντιστρόφως. Η ενναλαγή αυτή συμβαίνει κάθε ένα δευτερόλεπτο, με αποτέλεσμα η εφαρμογή μας να λειτουργεί και ως ένα τέλειο χρονόμετρο. Εφόσον από τη μια το λογισμικό σε

συνεργασία με το δέκτη λειτουργεί ως χρονόμετρο και από την άλλη είναι έτσι αναπτυγμένο ώστε να συνεργάζεται με το λογισμικό NTPD (Network Time Server Daemon) μπορεί να προσφέρει στο διαδίκτυο έναν UTC (Coordinated Universal Time) χρόνο ακρίβειας σε διάστημα του ενός δευτερολέπτου.

Για την αναπτύξη της εφαρμογής NORD χρησιμοποιήθηκε η αντικειμενοστραφής γλώσσα προγραματισμού C++ σε περιβάλλον Debian Linux. Στην επεξεργασία των δεδομένων που λαμβάνει ο δέκτης και στην δημιουργία των αρχείων RINEX χρησιμοποιήθηκε η βιβλιοθήκη ανοικτού κώδικα gpstk για ανάλυση δορυφορικών δεδομένων GPS που αναπτύχθηκε στο πανεπιστήμιο του Texas στο Austin των ΗΠΑ.

Πρόλογος

Καθ' όλη τη διάρκεια της εκπόνησης αυτής της διπλωματικής εργασίας μέχρι την ολοκλήρωσης της, εργαζόμουν ως επιστημονικός συνεργάτης σε ερευνητικά προγράμματα του Ιδρύματος Τεχνολογίας και Έρευνας, στο Εργαστήριο Γεωφυσικής-Δορυφορικής Τηλεπισκόπησης και Αρχαιοπεριβάλλοντος του Ινστιτούτου Μεσογειακών Σπουδών στο Ρέθυμνο. Υπήρξαν ορισμένα άτομα που με υποστήριξαν και με βοήθησαν και θέλω να εκφράσω τις προσωπικές μου ευχαριστίες.

Ευχαριστώ τον επιβλέποντα καθηγητή κ. Στέλιο Μερτίκα του Πολυτεχνείου Κρήτης γιατί με ευαισθητοποίησε στα θέματα Δορυφορικού Εντοπισμού και μου ανέθεσε αυτή την διπλωματική εργασία.

Ευχαριστώ τον κ. Kirill Palamartchouk για την πολύτιμη βοήθεια του και τις χρήσιμες συμβουλές του πάνω στην ανάπτυξη του λογισμικού και τις ιδιαιτερότητες των επιστημονικών οργάνων της Δορυφορικής Γεωδαισίας.

Ευχαριστώ τους καθηγητές κ. Κωνσταντίνο Καλαϊτζάκη και κ. Κωνσταντίνο Μπάλα του Πολυτεχνείου Κρήτης για τη συμμετοχή τους στην επιτροπή αξιολόγησης και την επιστημονική αξιολόγηση της εργασίας μου.

Ευχαριστώ την κοινότητα του ανοιχτού κώδικα στους διάφορους δικτυακούς τόπους (gpstk.sourceforge.net, http://gpsd.berlios.de, en.wikipedia.org) για τις προτάσεις και τις ερωτήσεις τους που με συνόδευαν στην ανάπτυξη του λογισμικού.

Τέλος, θα ήθελα να εκφράσω τις ευχαριστίες μου στον Αλέξανδρο Γιαννακίδη, συνάδελφο στο ΙΤΕ, για τις πολύτιμες παρατηρήσεις του και την Μαρία Γιαννουδάκη για τη φιλολογική επιμέλεια του κειμένου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ/ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

L1	Η πρώτη συχνότητα εκπομπής των δορυφορικών σημάτων 1575MHz
L2	Η δεύτερη συχνότητα εκπομπής των δορυφορικών σημάτων 1227MHz
P1	Μετρήσεις ψευδο-απόστασης στον κώδικα *P*, στην *L1* συχνότητα
P2	Μετρήσεις ψευδο-απόστασης στον κώδικα *P*, στην *L2* συχνότητα
C1	Μετρήσεις ψευδο-απόστασης στον κώδικα *C/A*, στην *L1* συχνότητα.
P	Οι μετρήσεις ψευδοαπόστασης
Φ	Η απόσταση δορυφόρου-δέκτη με μετρήσεις στη φάση σε μέτρα
φ	Η απόσταση δορυφόρου-δέκτη με μετρήσεις στη φάση σε κύκλους
λ	Το μήκος κύματος (λ_1= 0,1904m και λ_2=0,2443m)
c	Η ταχύτητα του φωτός
GPS	Global Positioning System
NORD	NovAtel OEM4 Receiver Daemon
RF	Radio Frequency
IF	Intermediate Frequency
ASCII	American Standard Code for Information Interchange
RTCA	Radio Technical Commission for Aviation Services
RTCN	Real Time Configuration Node
NMEA	National Marine Electronics Association
CRC	cyclic redundancy check
CR	Carriage Return
LF	Linefeed
EGNOS	European Geostationary Navigation Overlay Service
WAAS	Wide Area Augmentation System
SBAS	satellite-based augmentation system
DC	Direct Current
RTS	Request To Send

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχος της Εργασίας

Ο στόχος αυτής της Διπλωματικής Εργασίας είναι η ανάπτυξη ενός λογισμικού διαχείρισης και ελέγχου του δέκτη GPS OEM4 της εταιρίας NovAtel που βρίσκεται στο Εργαστήριο Γεωδαισίας και Γεωπληροφορικής στο Πολυτεχνείο Κρήτης. Την επικοινωνία με το δέκτη μέσω της σειριακής θύρας αναλαμβάνει ο υπολογιστής MARS που βρίσκετε και αυτός στο εργαστήριο. Ο σκοπός αυτού του σταθμού (MARS - GPS OEM4) θα είναι η παροχή δεδομένων του GPS με την υψηλότερη ακρίβεια, αξιοπιστία και ακεραιότητα.

Αρχικά το λογισμικό που έλεγχε το δέκτη ήταν αναπτυγμένο στη γλώσσα προγραμματισμού python. Η python, αν και μια πάρα πολύ δημοφιλής γλώσσα προγραμματισμού με πολλές βιβλιοθήκες και μεγάλη ευελιξία ανάπτυξης εφαρμογών, δεν κατάφερε να λύσει σημαντικά προβλήματα επικοινωνίας του υπολογιστή με το δέκτη όπως και της καταγραφής των αρχείων RINEX.

Η επιλογή της γλώσσας προγραμματισμού C++, με τη βοήθεια της οποίας αναπτύχθηκε το NORD, οφείλετε στη χρησιμοποίηση της βιβλιοθήκη ανοικτού κώδικα gpstk για ανάλυση δορυφορικών δεδομένων GPS, που αναπτύχθηκε στο Πανεπιστήμιο του Texas στο Austin των ΗΠΑ. Η βιβλιοθήκη gpstk είναι υλοποημένη σε C++ και αποτελεί ένα αξιόπιστο εργαλείο για την καταγραφή αρχείων παρατήρησης και πλοήγησης GPS σε μορφοποίηση RINEX.

Το λογισμικό NORD ελέγχου του δέκτη GPS OEM4 χρησιμοποιεί αρχιτεκτονική πολλαπλών επιπέδων (n-tier) διαχωρίζοντας έτσι τα στάδια υλοποίησης. Το λογισμικό αποτελείτε από δύο μέρη, αυτό του διακομιστή (server) και σε αυτό του πελάτη (client). Ο server δημιουργεί την διεπαφή μεταξύ του δέκτη και του υπολογιστή και ο client τη διεπαφή μεταξύ του χρήστη και του υπολογιστή.

Οι λειτουργίες που υποστηρίζονται από το λογισμικό NORD είναι οι παρακάτω:

- Διεπαφή χρήστη στο δέκτη (Web Interface – command line)
- Δημιουργία RINEX αρχείων (gpstk - Observation – Navigation Data)

- Network Time Server με τη χρήση του ntp
- Βαθμονόμηση του Δέκτη

1.2 Περιγραφή της Εργασίας

Η εργασία αυτή αναλύει αρχικά τη λειτουργία των δεκτών GPS OEM4 της εταιρίας NovAtel και την ανάπτυξη του λογισμικού διαχείρισής τους με την κωδική ονομασία NORD (NovAtel Oem4 Receiver Daemon). Στο επόμενο Κεφάλαιο αναπτύσσεται η θεωρία γύρω από την τεχνολογία δορυφορικού εντοπισμού και ναυσιπλοΐας GPS (Global Positioning System), όπου βασίζεται και η λειτουργία των δεκτών OEM4. Στο τρίτο Κεφάλαιο περιγράφεται η αρχιτεκτονική και οι δυνατότητες των δεκτών GPS OEM4 με ιδιαίτερη έμφαση στο κομμάτι των ηλεκτρονικών, των περιφερειακών και της λειτουργικότητας. Αναδεικνύεται η αξιοπιστία εντοπισμού και οι δυνατότητες παραμετροποίησης του συστήματος ανάλογα με τις απαιτήσεις των εφαρμογών. Στο τέταρτο Κεφάλαιο αναλύεται η μέθοδος ανάπτυξης του NORD για τη αποτελεσματική λήψη αποθήκευση δεδομένων. Επίσης εξηγείται αναλυτικά η δυνατότητα επικοινωνίας του NORD με εξωτερικούς και απομακρυσμένους χρήστες καθώς και η ευελιξία του περιβάλλοντος εργασίας.

Κεφάλαιο 2

Παγκόσμιο Σύστημα Εντοπισμού (GPS)

2.1 Εισαγωγή

Το Παγκόσμιο Σύστημα Εντοπισμού (Global Positioning System/ GPS) αναπτύχθηκε από το Υπουργείου Εθνικής Άμυνας των ΗΠΑ και τέθηκε σε πλήρη λειτουργία το 1994. Ονομάζεται επίσης και NAVSTAR GPS (**NAV**igation **S**atellite **T**iming **A**nd **R**anging). Χρησιμοποιείται ευρέως στην πλοήγηση, στα Γεωγραφικά Συστήματα Πληροφοριών, στην επιστημονική έρευνα και σε πολλές άλλες δραστηριότητες που έχουν εφαρμογές με τον προσδιορισμό θέσης στην επιφάνεια της Γης.

Οι μετρήσεις του GPS γίνονται ως εξής: ο δέκτης λαμβάνει δορυφορικά σήματα και αποκωδικοποιώντας τα καθορίζει την απόσταση δορυφόρου – δέκτη καθώς και την ταχύτητα μεταβολής της. Ο ακριβής καθορισμός της απόστασης επιτυγχάνεται με το να υπολογίζεται από το δέκτη ο χρόνος μετάδοσης του σήματος από το δορυφόρο στο δέκτη με μεγάλη ακρίβεια. Τελικά, η απόσταση αυτή προκύπτει από τον πολλαπλασιασμό του χρόνου μετάδοσης του σήματος με την ταχύτητα διάδοσης του ηλεκτρομαγνητικού κύματος (ταχύτητα του φωτός).

Το GPS αποτελείται από τρία τμήματα. Το δορυφορικό τμήμα (space segment), το τμήμα ελέγχου (control segment) και το τμήμα χρήσης (user segment).

Το δορυφορικό τμήμα (space segment) αποτελούνταν αρχικά από 21 δορυφόρους και τρεις αναπληρωματικούς, σε έξι τροχιακά επίπεδα ύψους 20.000km πάνω από την επιφάνεια της Γης. Κάθε τροχιακό επίπεδο έχει κλίση 55^o ως προς τον ισημερινό, σχηματίζει γωνία 60^o με το επόμενο τροχιακό επίπεδο στο επίπεδο του ισημερινού και περιέχει τέσσερις δορυφόρους κατανεμημένους στο χώρο σε γωνίες 120^o. Ο παραπάνω τροχιακός σχεδιασμός παρέχει πλήρη κάλυψη (6 τουλάχιστον δορυφόροι ορατοί σε ένα τόπο όλη την ημέρα). Από το Σεπτέμβριο του 2007 υπάρχουν 31 δορυφόροι GPS σε λειτουργία.

Το τμήμα ελέγχου αποτελείται από πέντε σταθμούς ελέγχου και παρακολούθησης των δορυφορικών σημάτων και ένα κύριο σταθμό ελέγχου (Master Control Station) που βρίσκεται στο Colorado Springs των ΗΠΑ. Οι συντεταγμένες των σταθμών παρακολούθησης είναι γνωστές με πολύ μεγάλη ακρίβεια και έτσι προσδιορίζονται οι τροχιές των δορυφόρων. Στη συνέχεια οι πληροφορίες αυτές στέλνονται στον κύριο σταθμό ελέγχου με συνέπεια να προβλέπονται οι δορυφορικές τροχιές και τα σφάλματα των δορυφορικών χρονομέτρων.

Το τμήμα χρήσης περιλαμβάνει τους δέκτες GPS που φέρουν πλοία, αεροσκάφη, οχήματα και αλλού οι οποίοι λαμβάνουν, επεξεργάζονται τα σήματα και καταγράφουν τις μετρήσεις. Ο δέκτης αποτελείται από την κεραία, τον κυρίως δέκτη και τον υπολογιστή.

2.2 Αρχή λειτουργίας του GPS

Στο δέκτη GPS γίνεται η λήψη και η ανάλυση του λαμβανομένου σήματος και μέσω μετρήσεων αποστάσεων μεταξύ δορυφόρων-δέκτη, προσδιορίζεται η θέση του δέκτη. Επειδή οι δέκτες GPS διαθέτουν κατά κανόνα χρονόμετρα χαμηλής ή μέσης ακρίβειας και όχι ατομικά χρονόμετρα (ρουβιδίου ή καισίου) όπως οι δορυφόροι του συστήματος, εκτός των ατμοσφαιρικών χρονικών καθυστερήσεων έχουμε και τις χρονικές καθυστερήσεις που οφείλονται κυρίως στο χρονόμετρο του δέκτη, αλλά και δευτερευόντως, του δορυφόρου. Έτσι, κατά τον προσδιορισμό θέσης ενός δέκτη, στις άγνωστες ποσότητες εκτός από τις τρεις συντεταγμένες του δέκτη (X,Y,Z ή φ,λ,h) προστίθεται και ένας επιπλέον άγνωστος dT, που αντιπροσωπεύει τη χρονική καθυστέρηση του χρονομέτρου του δέκτη σε σχέση με το χρόνο αναφοράς του GPS. Ο χρόνος αναφοράς του GPS έχει έναρξη την 0h U.T.C. της 5ης Ιανουαρίου του 1980. Η προσδιοριζόμενη θέση (X,Y,Z) αναφέρεται στο Παγκόσμιο Γεωκεντρικό Σύστημα Αναφοράς 1984, γνωστό ως WGS84. Οι μετρήσεις του GPS διακρίνονται σε δύο βασικές κατηγορίες: σε μετρήσεις ψευδοαποστάσεων (pseudoranges) και σε μετρήσεις φάσεων (phase measurements). Από αυτές τις δύο ακριβέστερες είναι οι μετρήσεις φάσεων. Υπάρχουν γενικά δύο μέθοδοι προσδιορισμού θέσης, η στατική και η κινηματική. Στο στατικό προσδιορισμό ο δέκτης είναι στάσιμος και οι παρατηρήσεις διαρκούν από λίγα λεπτά μέχρι μερικές ώρες, ενώ στον κινηματικό ο

δέκτης βρίσκεται σε κίνηση λαμβάνοντας συνεχώς το δορυφορικό σήμα. Ο τρόπος προσδιορισμού θέσης με GPS μπορεί να είναι απόλυτος (absolute positioning), ή σχετικός (relative positioning). Στον απόλυτο εντοπισμό η θέση του δέκτη (X,Y,Z) υπολογίζεται ως προς το γεωκεντρικό σύστημα αναφοράς ενώ στο σχετικό η θέση του δέκτη καθορίζεται σε σχέση με κάποιο άλλο δέκτη (ΔX,ΔY,ΔZ). Στο σχετικό εντοπισμό αντί των πρωτογενών παρατηρήσεων είναι δυνατό να χρησιμοποιηθούν οι λεγόμενες διαφορές (ψευδοαποστάσεων, φάσεων), συνήθως απλές, διπλές ή και τριπλές διαφορές. Στις εφαρμογές με απαιτήσεις μεγάλης ακριβείας, π.χ. αποτυπώσεις σε μεγάλες κλίμακες, γεωδαιτικά δίκτυα κάθε είδους, χρησιμοποιούνται οι τεχνικές του σχετικού προσδιορισμού (διαφορικός εντοπισμός - differential positioning).

2.3 Σήματα Ναυσιπλοίας

Οι δορυφόροι του συστήματος GPS εκπέμπουν ηλεκτρομαγνητικά κύματα σε δύο φέρουσες συχνότητες (carrier frequencies), την L1 και L2, που παράγονται από την θεμελιώδη συχνότητα των 10,23 MHz:

L1 : 154 x 10,23 MHZ = 1575,42 MHz ($\lambda_1 \approx 19,05$ cm)
L2 : 120 x 10,23 MHZ = 1227,60 MHz ($\lambda_2 \approx 24,45$ cm)

Η διαμόρφωση των φερουσών συχνοτήτων L1 και L2 (εκπεμπόμενων σημάτων) γίνεται με τον λεγόμενο κώδικα PRN (Pseudo Random Noise Code), ο οποίος αποτελείται από μια σειρά από +1 και -1 που μοιάζει τυχαία. Συγκεκριμένα υπάρχουν διαθέσιμοι τρεις κώδικες, οι P, C/A και D.

α) Ο πρώτος κώδικας ονομάζεται P – ακριβής κώδικας – (Precision code). Έχει ταχύτητα μεταβολής των κωδίκων (bits) ίση με την θεμελιώδη συχνότητα f=10,23 MHz του GPS, μήκος παλμού ≈ 30 m και επαναλαμβάνεται κάθε περίπου 267 ημέρες. Ο κώδικας P μεταδίδεται τόσο με την L1 όσο και με την L2 συχνότητα. Ένα τμήμα του κώδικα P, διάρκειας 7 ημερών, είναι αποθηκευμένο στη μνήμη του κάθε δορυφόρου και μεταδίδεται αποκλειστικά. Το τμήμα αυτό επανεκκινείται τα

μεσάνυχτα του Σαββάτου προς Κυριακή 0h UT κάθε βδομάδα. Στην περίπτωση που έχουμε εσκεμμένη παρέμβαση στα σήματα του GPS για μείωση της ακριβείας στην κατάσταση «μη-παρεμβολής» (Anti-Spoofing, AS: προστασία εναντίον πλαστών σημάτων – βλέπε παρακάτω) ο κώδικας P πολλαπλασιάζεται με έναν άγνωστο κώδικα W και μετατρέπεται σε ένα κρυπτογραφημένο κώδικα Y, στον οποίο έχουν πρόσβαση μόνο εξουσιοδοτημένοι χρήστες.

β) Ο δεύτερος κώδικας ονομάζεται κώδικας C/A (Coarse/acquisition code). Είναι χαμηλότερης ακρίβειας από τον P (10 φορές χαμηλότερη), έχει ταχύτητα μεταβολής των κωδίκων f=1,023 MHz και επαναλαμβάνεται κάθε 1 msec. Διαμορφώνεται και κατά συνέπεια μεταδίδεται μόνο στη συχνότητα L1.

γ) Ο τρίτος κώδικας ονομάζεται κώδικας D (Data code) ή και μήνυμα ναυσιπλοΐας (navigation message). Αποτελείται από μια σειρά από bits και έχει συχνότητα 50Hz ενώ παρέχει πληροφορίες για το χρόνο εκπομπής του σήματος από το δορυφόρο σε κάθε χρονική στιγμή, για τις διορθώσεις στις ατμοσφαιρικές καθυστερήσεις των δορυφορικών χρονομέτρων, για τα στοιχεία τροχιάς των δορυφόρων κ.λ.π. Το κάθε ναυτιλιακό μήνυμα περιέχει πληροφορία μεγέθους 1500 bits, συνολικής διάρκειας 30 sec, και ρυθμό μετάδοσης 50 bps. Το μήνυμα περιέχει πέντε τμήματα μεγέθους 300 bits το καθένα, διάρκειας 5 sec το καθένα κατανεμημένα σε τρία block. Το κάθε τμήμα έχει 10 «λέξεις» των 30 bits από τα οποία τα έξι bits είναι σήματα ελέγχου. Οι δύο πρώτες «λέξεις» κάθε τμήματος του ναυτιλιακού μηνλυματος είναι οι: α) TLM (Telemetry) που περιέχει στοιχεία για το συγχρονισμό με τον οποίο διευκολύνεται η πρόσβαση στα στοιχεία της ναυσιπλοΐας και β) HOW (Hand Over Word) που περιέχει στοιχεία για τον κώδικα P και είναι απαραίτητο για τον άμεσο εντοπισμό του μέρους του P κώδικα που λαμβάνει ο δέκτης. Η δομή των τριών block που περιλαμβάνονται στο σήμα ναυσιπλοΐας περιλαμβάνουν:

1. Το block I περιλαμβάνει το πρώτο τμήμα του ναυτιλιακού μηνύματος και περιέχει πληροφορίες για τον τρόπο περιγραφής του χρονομέτρου του δορυφόρου.
2. Το block II περιλαμβάνει τα τμήματα 2 και 3 και περιέχει τις παραμέτρους που χρησιμεύουν για τον υπολογισμό των τροχιών των δορυφόρων.
3. Το block III περιλαμβάνει τα τμήματα 4 και 5 και περιέχει στοιχεία για την συμπεριφορά του χρονομέτρου, τις εφημερίδες όλων των δορυφόρων του συστήματος GPS και στοιχεία για την ιονοσφαιρική διόρθωση. Το μεγαλύτερο τμήμα

του αφορά πληροφορίες για εξουσιοδοτημένους χρήστες. Κάθε Σάββατο τα μεσάνυχτα όλοι οι κώδικες επαναλαμβάνονται από την αρχή. Για κάθε δορυφόρο αντιστοιχεί επίσης μία μοναδική δομή του κώδικα P. Πρέπει να σημειώσουμε εδώ ότι ο ακριβής κώδικας P δεν μπορεί να χρησιμοποιηθεί για πολιτικούς σκοπούς σε συνεχή βάση. Από την αρχή λειτουργίας του το GPS χρησιμοποιήθηκε για στρατιωτικούς σκοπούς, με αποτέλεσμα, ελάχιστες ήταν οι φορές που ο ακριβής κώδικας P αφηνόταν σε ελεύθερη πολιτική χρήση.

Χάριν συντομίας στην συνέχεια της εργασίας ορίζονται οι παράμετροι:

- *P1*: Μετρήσεις ψευδο-αποστάσεων στον κώδικα *P*, στην συχνότητα *L1*.
- *P2*: Μετρήσεις ψευδο-αποστάσεων στον κώδικα *P*, στην συχνότητα *L2*.
- *C1*: Μετρήσεις ψευδο-αποστάσεων στον κώδικα *C/A*, στην συχνότητα *L1*.
- Φ1, Φ2: Μετρήσεις φάσης αποστάσεων στις συχνότητες *L1* και *L2* αντίστοιχα σε μονάδες μήκους.

2.4 Ακρίβεια Εντοπισμού

Η ακρίβεια εντοπισμού ενός σημείου με τη χρήση ενός δέκτη GPS εξαρτάται από φυσικούς παράγοντες, όπως η σκίαση των δεκτών από εμπόδια, η τροποσφαιρική και η ιονοσφαιρική καθυστέρηση, πολλαπλές ανακλάσεις εδάφους, η επιλεκτική διάθεση και η «απόκρυψη» του κώδικα P (anti-spoofing).

Η **επιλεκτική διάθεση του σήματος** στηρίζεται σε δύο μεθόδους παραποίησης των πληροφοριών της μεταδιδόμενης εφημερίδας, στην πρώτη μέθοδο προστίθενται σφάλματα στις παραμέτρους του χρονομέτρου του δορυφόρου, και στη δεύτερη που προστίθενται σφάλματα στις παραμέτρους της τροχιάς του δορυφόρου. Με τον τρόπο αυτό, η αρχική ακρίβεια των 15-40 m που προκύπτει από ψευδοαποστάσεις μέσω του κώδικα C/A, μειώνεται σε ≈100 m (φ,λ) και ≈150 m (h). Η μέθοδος της επιλεκτικής διαθεσιμότητας για τη μείωση της ακρίβειας απενεργοποιήθηκε την 1^{η} Μαΐου 2002 οπότε και η αρχική ακρίβεια που προσφέρει το GPS στον προσδιορισμό θέσης βελτιώθηκε.

Η μέθοδος **Anti-Spoofing** "αποκρύπτει" τον κώδικα P, πολλαπλασιάζοντάς τον με κάποιο μυστικό κώδικα W. Αποτέλεσμα του παραπάνω συνδυασμού είναι η εμφάνιση ενός κρυφού κώδικα Y. Συνεπώς ο μη εξουσιοδοτημένος χρήστης δεν μπορεί να έχει πρόσβαση στον κώδικα P με αποτελέσμα εσφαλμένους υπολογισμούς. Η παραμόρφωση λόγω του Anti-Spoofing ενεργοποιήθηκε στις 00:00 UTC στις 31 Ιανουαρίου 1994 από τις στρατιωτικές υπηρεσίες των Η.Π.Α. και παραμένει μέχρι σήμερα.

2.5 Μετρήσεις με το GPS

Το GPS μετράει τις αποστάσεις μεταξύ των δορυφόρων και του δέκτη που προκύπτουν από τις μετρήσεις της διαφοράς στο χρόνο ή στη φάση του εκπεμπόμενου και του λαμβανόμενου σήματος. Με αυτόν τον τρόπο παράγονται οι μετρήσεις ψευδοαποστάσεων (pseudoranges) στον κώδικα (*C1*, *P1* και *P2*) και οι μετρήσεις φάσης (Φ1, Φ2) του φέροντος κύματος. Επιπλέον καταγράφεται και η ολοκλήρωση των μετρήσεων της φάσης σε δεδομένο χρονικό διάστημα, ή αλλιώς μετρήσεις Doppler.

2.5.1 Μετρήσεις ψευδοαποστάσεων στον κώδικα

Στην Ενότητα 2.2 αναφέρθηκε ότι ο προσδιορισμός της απόστασης δορυφόρου-δέκτη γίνεται με την ακριβή μέτρηση του χρόνου μετάδοσης του σήματος από τον δορυφόρο στον δέκτη. Αυτό επιτυγχάνεται με τη συσχέτιση του κώδικα του εισερχόμενου σήματος με ένα ακριβές αντίγραφο του κώδικα που παράγεται στο δέκτη. Επειδή και οι δύο μορφές του κώδικα αναφέρονται στο ίδιο χρονικό σύστημα η χρονική καθυστέρηση που απαιτείται για τη μέγιστη συσχέτιση είναι ένα μέτρο του χρόνου μετάδοσης. Αν πολλαπλασιαστεί ο χρόνος αυτός με την ταχύτητα διάδοσης του κύματος (ταχύτητα του φωτός) καθορίζεται η απόσταση δορυφόρου-δέκτη. Επειδή τα χρονόμετρα του δορυφόρου και του δέκτη υπόκεινται σε σφάλματα, η μετρούμενη απόσταση καλείται *ψευδοαπόσταση* (pseudorange). Η εξίσωση παρατήρησης της ψευδοαπόστασης μεταξύ δορυφόρου-δέκτη είναι:

$$P = \rho + d\rho + c(dt - dT) + d_{ion} + d_{trop} + m_p + \varepsilon_p \quad (2.1)$$

όπου,

P: ψευδοαπόσταση σε μονάδες μήκους (m)

ρ: γεωμετρική απόσταση δορυφόρου – δέκτη (m)

$d\rho$: σφάλμα στην απόσταση εξαιτίας λανθασμένων δορυφορικών εφημερίδων

c: ταχύτητα του φωτός (m / sec)

dt: σφάλμα του δορυφορικού χρονομέτρου (sec)

dT: σφάλμα του χρονομέτρου του δέκτη (sec)

d_{ion}: σφάλμα εξαιτίας της ιονόσφαιρας (m)

d_{trop}: σφάλμα εξαιτίας της τροπόσφαιρας (m)

m_p: σφάλμα εξαιτίας πολυκλαδικών ανακλάσεων (m)

ε_p : τυχαία σφάλματα (m)

Η γεωμετρική απόσταση ρ μεταξύ δορυφόρου-δέκτη δίνεται από τη σχέση:

$$\rho = \left\| r_i - R \right\| = \sqrt{(x_i - X)^2 + (y_i - Y)^2 (z_i - Z)^2} \quad (2.2)$$

όπου,

$r(x_i, y_i, z_i)$: διάνυσμα θέσης του δορυφόρου

$R(X, Y, Z)$: διάνυσμα θέσης του δέκτη

2.5.2 Μετρήσεις ψευδοαποστάσεων στη φάση

Διάφοροι δέκτες του GPS μετρούν τη διαφορά φάσης μεταξύ του σήματος που παράγεται από τον δέκτη και του λαμβανόμενου δορυφορικού σήματος. Αυτό γίνεται όταν ο κώδικας έχει εξαλειφθεί από το φέρον κύμα. Η εξίσωση παρατήρησης της ψευδοαπόστασης μεταξύ δορυφόρου-δέκτη στη μέτρηση της φάσης είναι:

$$\Phi = -\lambda\phi = \rho + d\rho + c(dt - dT) + \lambda N - d_{ion} + d_{trop} + m_p + \varepsilon_\phi \quad (2.3)$$

Φ : η απόσταση (m)

λ : το μήκος κύματος (λ_1= 0,1904m και λ_2=0,2443m)

ϕ : ο αριθμός κύκλων του σήματος

N : η αβεβαιότητα κύκλων (ακέραιος αριθμός κύκλων).

m_p : το σφάλμα εξαιτίας των πολυκλαδικών ανακλάσεων (m)

ε_ϕ : τα τυχαία σφάλματα (m)

Οι μετρήσεις φάσης είναι ένα μέτρο του αριθμού των κύκλων που έχουν περάσει από τη στιγμή που ο δέκτης «κλειδώσει» (locked) στο δορυφορικό σήμα συν κάποιο τυχαίο αριθμό κύκλων. Έτσι η εξίσωση (2.3) είναι παρόμοια με την (2.1) εκτός από τον όρο N που αντιπροσωπεύει την αβεβαιότητα κύκλων. Ο N είναι ο άγνωστος ακέραιος αριθμός των κύκλων μεταξύ του δορυφόρου και του δέκτη τη στιγμή που ο δέκτης κλείδωσε το σήμα.

2.5.3 Μετρήσεις Doppler

Ο τρίτος τρόπος μέτρησης είναι η παρατήρηση του φαινομένου Doppler, ο υπολογισμός δηλαδή της στιγμιαίας μετάθεσης της συχνότητας του εισερχόμενου GPS σήματος. Αυτή η μετάθεση οφείλεται στην κίνηση του δορυφόρου σε σχέση με τον δέκτη και είναι ένα μέτρο της μεταβολής της μεταξύ τους απόστασης. Πρέπει να επισημανθεί ότι οι μετρήσεις Doppler δεν είναι διαφορετικές μετρήσεις αλλά ολοκληρωμένες μετρήσεις φάσης σε δεδομένο χρονικό διάστημα. Η σχετική εξίσωση είναι της μορφής:

$$D = d(\rho)/dt + d[d\rho - d_{ion} + d_{trop} + c(dt - dT) + m_p]/dt + \varepsilon_D \qquad (2.3)$$

όπου ,

D : ο στιγμιαίος ρυθμός μεταβολής της απόστασης (m/ sec)

ε_D : τα τυχαία σφάλματα (m/ sec)

2.6 Δέκτες GPS

Οι δέκτες του συστήματος GPS λαμβάνουν τα σήματα των δορυφόρων και στη συνέχεια υπολογίζουν την απόσταση μεταξύ δορυφόρου – δέκτη είτε με τη μέτρηση ψευδοαποστάσεως (χρήση κώδικα) είτε με τη μέτρηση της διαφοράς φάσης της φέρουσας συχνότητας μεταξύ δορυφόρου δέκτη, δηλαδή ανάλογα με τη μέθοδο μέτρησης που θα χρησιμοποιήσουμε.

Οι δέκτες του GPS αποτελούνται από τα εξής μέρη (Seeber, 1993):

- Την κεραία που μετατρέπει το ηλεκτρομαγνητικό κύμα που εκπέμπεται από τον δορυφόρο σε ηλεκτρικό ρεύμα που μπορεί να επεξεργαστεί από τα ηλεκτρονικά συστήματα του δέκτη.
- Τη μονάδα ραδιοσυχνότητας που συνδυάζει το εισερχόμενο σήμα με ένα ημιτονικό σήμα που δημιουργεί ο ταλαντωτής του δέκτη για την μετατροπή της συχνότητας του εισερχόμενου σήματος στη μικρότερη δυνατή.
- Τον μικροεπεξεργαστή για την επεξεργασία του δορυφορικού σήματος και την μονάδα αποκωδικοποίησης του δορυφορικού μηνύματος.
- Τον ταλαντωτή που παράγει το ημιτονικό σήμα.
- Τη μονάδα παροχής ισχύος.
- Το σύστημα καταγραφής των στοιχείων.
- Το σύστημα εμφάνισης των στοιχείων.

Γενικά υπάρχουν δύο κύριες κατηγορίες δεκτών GPS. Οι *γεωδαιτικοί* (geodetic receivers) και οι δέκτες ναυσιπλοίας (navigation receivers). Οι γεωδαιτικοί δέκτες παρέχουν όλα τα είδη μετρήσεων του GPS (Φ1, Φ2, *C1*, *P1* και *P2*), είναι σχετικά ακριβοί (15.000-20.000 δολάρια ΗΠΑ) και χρησιμοποιούνται για εφαρμογές που απαιτούν υψηλή ακρίβεια εντοπισμού (μελέτη μικρομετακινήσεων, μελέτη σεισμικά ενεργών περιοχών, προσγείωση αεροσκαφών, γεωδαιτικές εφαρμογές). Οι δέκτες ναυσιπλοίας είναι συνήθως μονής συχνότητας (*C1*, *P1*, Φ1), είναι αρκετά φθηνότεροι (500-5.000 δολάρια ΗΠΑ) και παρέχουν χαμηλότερη ακρίβεια. Χρησιμοποιούνται κυρίως σε εφαρμογές πλοήγησης σκαφών και οχημάτων, καθώς και σε γεωδαιτικές

εφαρμογές μικρής κλίμακας. Οι σύγχρονοι δέκτες GPS έχουν την δυνατότητα εντοπισμού ακόμα και σε λειτουργία *anti-spoofing*.

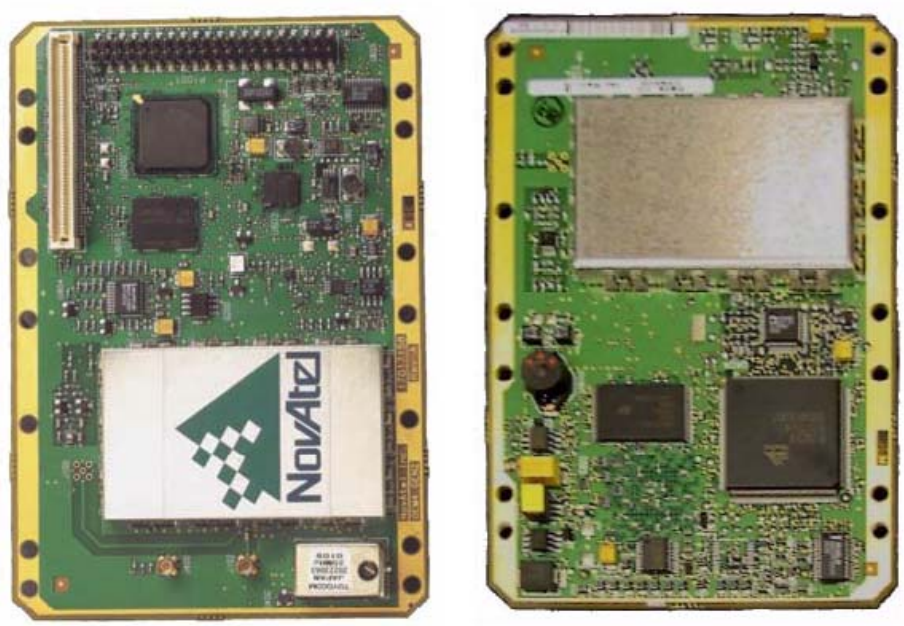
Η σειρά OEM4 καρτών GPS της NovAtel χρησιμοποιείται σε γεωδαιτικούς σταθμούς σαν αυτό που εδρεύει στο Εργαστήριο Γεωδαισίας και Γεωπληροφορικής στο Πολυτεχνείο Κρήτης και πάνω στον οποίο ανάπτύχθηκε το λογισμικό NORD. Βασικά χαρακτηριστικά της σειράς OEM4 είναι το μικρό μέγεθος της κάρτας (10cm x 10cm, και ζυγίζει μόλις 56 γραμμάρια), οι δύο σειριακές θύρες επικοινωνίας και μία USB , τα 24 κανάλια παράλληλης παρακολούθησης δορυφόρων GPS και άλλα τεχνικά χαρακτηριστικά που θα περιγραφούν αναλυτικά στο επόμενο κεφάλαιο.

Κεφάλαιο 3

Τεχνικά χαρακτηριστικά καρτών GPS NovAtel της σειράς OEM4

3.1 Εισαγωγή

Η σειρά OEM4 είναι ένα σύνολο ισχυρών, παραμετροποιήσιμων καρτών GPS, ικανών για τη λήψη και παρατήρηση *L1 C/A* κώδικα, *L1 και L2* σημάτων και *L2 P* κώδικα (ή τον κρυπτογραφημένο κώδικα Y) έως και 12 δορυφόρων. Είναι ικανές να λειτουργήσουν σε περιοχές με υψηλά ηλεκτρομαγνητικά δυναμικά και με συχνές διακοπές των σημάτων. Διαθέτουν επεξεργαστές 32-bit και μια σειρά διαφορετικών θυρών εισόδου-εξόδου (σειριακών και USB) .



Σχήμα 3.1: OEM4-G2 GPSCard

Τα κύρια χαρακτηριστικά της σειράς OEM4 είναι τα εξής:

- 24 κανάλια παράλληλης παρακολούθησης δορυφόρων
- Γρήγορη επανάκτηση του σήματος των δορυφόρων

- Πλήρως αναβαθμίσιμο λογισμικό
- Χαμηλή κατανάλωση ενέργειας
- 20Hz συχνότητα μεταφοράς πρωτογενών δεδομένων (20Hz raw data output).
- Παρακολούθηση σημάτων στις συχνότητες L1/L2
- Υποστήριξη για SBAS διορθώσεις, όπως αυτές που δημιουργούνται από το WAAS και EGNOS συστήματα

3.2 Κάρτες GPS OEM4

Πρόκειται ουσιαστικά για μια ηλεκτρονική πλακέτα με ενσωματωμένη ραδιοσυχνότητα (RF). Η βασική συστοιχία λειτουργίας της αποτελείται από μια πηγή τροφοδοσίας, μια κεραία GPS και ένα σύστημα λήψης ή μετάδοσης δεδομένων. Τα δύο βασικά μοντέλα είναι οι κάρτες OEM4-G2 και OEM4-G2L, που εκτός από κάποιες διαφορές που αφορουν στις θύρες επικοινωνίας , στην κατανάλωση ενέργειας και στο μέγεθος, είναι πανομοιότυπες ως προς την επεξεργαστική τους ικανότητα. Γι' αυτό το λόγο εξετάζονται και παρουσιάζονται ως μια σειρά καρτών.
Το κύρια στοιχεία των καρτών GPS OEM4 είναι το *τμήμα ραδιοσυχνότητας (RF) και το τμήμα ψηφιακών κυκλωμάτων.*

3.2.1 Τμήμα Ραδιοσυχνότητας (RF)

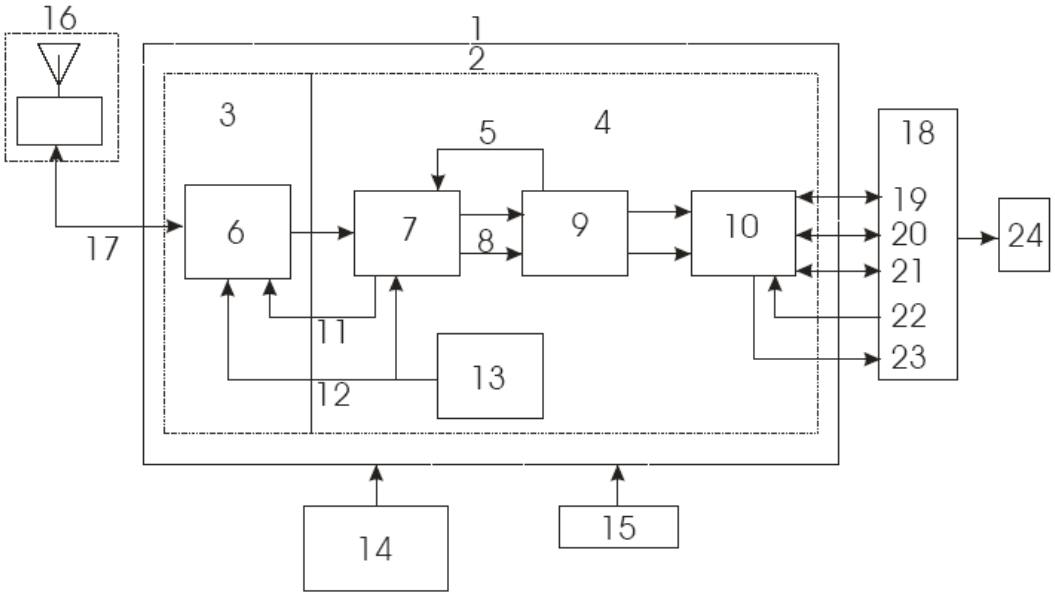
Ο δέκτης λαμβάνει ένα φιλτραρισμένο και ενισχυμένο σήμα GPS από την κεραία μέσω ομοαξονικού καλωδίου. Το *τμήμα ραδιοσυχνότητας (RF)* μεταφράζει το εισερχόμενο σήμα RF σε ένα IF (intermediate frequency) σήμα το οποίο χρησιμοποιείται από το *τμήμα ψηφιακών κυκλωμάτων*. Παρέχει επίσης την ισχύ στο LNA (Low Noise Amplifier) της ενεργού κεραίας μέσω του ομοαξονικού καλωδίου, διατηρώντας μονωμένα τα κυκλωμάτα DC (συνεχούς ρεύματος) και RF. Το τμήμα RF μπορεί να απορρίψει ένα υψηλό επίπεδο μιας πιθανής παρεμβολής, π.χ., MSAT (Mobile telephony Satellite), Inmarsat, κυψελοειδές τηλέφωνο, και sub-harmonic σήματα TV.

3.2.2 Τμήμα ψηφιακών κυκλωμάτων

Το ψηφιακό τμήμα του δέκτη λαμβάνει ένα ενισχυμένο σήμα GPS το οποίο το ψηφιοποιεί και το επεξεργάζεται κατάλληλα για να βρει μια λύση GPS (θέση, ταχύτητα και χρόνος). Το ψηφιακό τμήμα αποτελείται από έναν μετατροπέα A/D (αναλογικό σε ψηφιακό), έναν επεξεργαστή 32bit, μια μνήμη, κυκλώματα επεξεργασίας σήματος, τις περιφερειακές συσκευές, τα βοηθητικά στοιχεία του κυκλώματος και διέπεται από μια παραμετροποιήσιμη λογική ελέγχου.
Το ψηφιακό τμήμα εκτελεί τους απαραίτητους υπολογισμούς για να μετατρέψει τα αναλογικά σήματα IF σε αξιοποιήσιμα δεδομένα για τη θέση και κατάσταση των δορυφόρων. Διαχειρίζεται επίσης όλες τις λειτουργίες εισόδου – εξόδου (serial ports, USB) καθώς επίσης και τη λειτουργία της κεραίας.

3.2.3 Κεραίες GPS

Ο σκοπός της GPS κεραίας είναι να μετατραπούν τα ηλεκτρομαγνητικά κύματα που διαβιβάζονται από τους GPS δορυφόρους σε σήματα RF. Μια ενεργή GPS κεραία απαιτείται για να λειτουργήσει ο δέκτης κατάλληλα. Η ηλεκτρική ενέργεια που απαιτείται για την κανονική λειτουργία της κεραίας παρέχεται κανονικά από το δέκτη. Εντούτοις, εάν χρησιμοποιηθεί ένας διαφορετικός τύπος κεραίας που παρουσιάζει ασυμβατότητα με αυτή την τροφοδοσία, πρέπει να συνδεθεί με μια εξωτερική πηγή ενέργειας, δηλαδή είτε με κάποια γεννήτρια ρεύματος, είτε με ένα δίκτυο ηλεκτροδότησης.



Σχήμα 3.2: Λειτουργικό διάγραμμα GPS δέκτη.

#	Περιγραφή	#	Περιγραφή
1	Συσκευασία	13	VCTCXO
2	GPS κάρτα	14	Προαιρετική Ενέργεια LNA
3	Τμήμα RF	15	Ενέργεια
4	Ψηφιακό Τμήμα	16	Κεραία GPS και LNA
5	Έλεγχος	17	RF και Ενέργεια
6	RF-IF Τμήμα	18	Επεξεργαστής Δεδομένων και Σήματος
7	Επεξεργαστής Σήματος	19	COM1
8	Ρολόι	20	COM2
9	22-Bit CPU	21	COM3
10	Σύστημα Εισόδου / Εξόδου	22	Χρονικό Σήμα Εισόδου
11	AGC	23	Χρονικό Σήμα Εξόδου
12	Ρολόι	24	Επικοινωνία USB

Πίνακας 3.1: Υπόμνημα Σχημάτος 3.2

3.3 Επικοινωνία χρήστη με το δέκτη

Η επικοινωνία με το δέκτη βασίζεται στην αποστολή εντολών στο δέκτη μέσω της σειριακής θύρας από μια εξωτερική συσκευή. Αυτή θα μπορούσε να είναι είτε ένα τερματικό είτε ένας συμβατός ηλεκτρονικός υπολογιστής που συνδέεται άμεσα στη σειριακή θύρα επικοινωνίας του δέκτη χρησιμοποιώντας ένα σειριακό καλώδιο RS232.

3.3.1 Προεπιλεγμένες ρυθμίσεις της σειριακής θύρας

Ο δέκτης επικοινωνεί με τον ηλεκτρονικό υπολογιστή ή το τερματικό μέσω της σειριακής θύρας. Για να συμβεί αυτό, η σειριακή θύρα του δέκτη και του χειριστή πρέπει να διαμορφωθούν καταλλήλως. Οι σειριακές θύρες COM1,COM2 και COM3 του δέκτη έχουν τις ακόλουθες προεπιλεγμένες ρυθμίσεις:

- 9600 bps, no parity, 8 data bits, 1 stop bit, no handshaking, echo off

Για την αλλαγή των προεπιλεγμένων ρυθμίσεων απαιτείται να δοθεί η εντολή COM η οποία έχει τη παρακάτω σύνταξη:

COM [port] bps parity][databits[stopbits[handshake[echo[break]]]]]

Η ταχύτητα μεταφοράς δεδομένων που επιλέγεται καθορίζει το πόσο γρήγορα διαβιβάζονται οι πληροφορίες από το δέκτη στον υπολογιστή. Η πραγματική ταχύτητα μεταφοράς των δεδομένων εξαρτάται από τον αριθμό των δορυφόρων που παρακολουθεί ο δέκτης, τα φίλτρα που χρησιμοποιούνται και τον χρόνο αδράνειάς του (idle time).

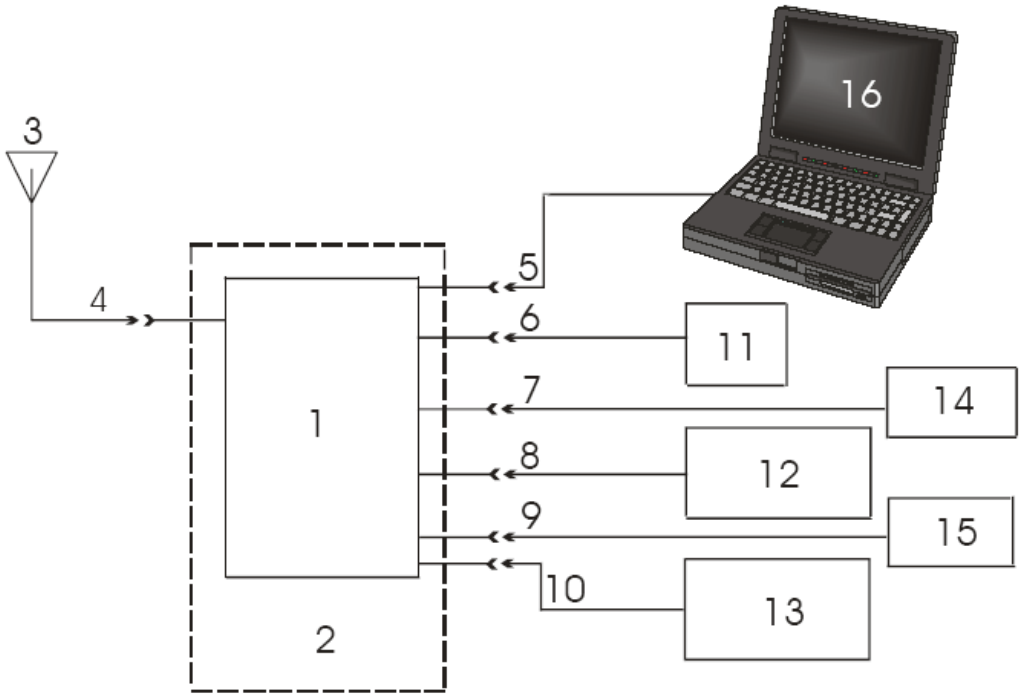
3.3.2 Επικοινωνία με απομακρυσμένο τερματικό

Μια μέθοδος επικοινωνίας με το δέκτη είναι μέσω ενός απομακρυσμένου τερματικού. Ο δέκτης GPS συνδέεται με σειριακό καλώδιο στην κατάλληλη RS232 διεπαφή με το

τερματικό. Για να επικοινωνήσει με το τερματικό ο δέκτης, απαιτείται να χρησιμοποιηθούν μόνο οι ακροδέκτες λήψης, εκπομπής και γείωσης της σειριακής θύρας επικοινωνίας RX(Receive), TX(Transmitt) και GND(γείωση) αντίστοιχα Το Handshaking δεν απαιτείται, αν και μπορεί προαιρετικά να χρησιμοποιηθεί.

3.3.3 Επικοινωνία με ηλεκτρονικό υπολογιστή

Ένας ηλεκτρονικός υπολογιστής σε συνεργασία με ένα λογισμικό, σαν αυτό που αναπτύχθηκε (NORD), μπορεί να προσθέσει μεγάλη ευελιξία στη λήψη, επεξεργασία και αποθήκευση των δεδομένων GPS. Όλα τα δεδομένα στέλνονται από το δέκτη σε πρωτογενή μορφή (raw data) μέσω της σειριακής θύρας επικοινωνίας και οι εντολές λαμβάνονται σε μορφή συμβολοσειρών (strings).



Σχήμα 3.3: Επικοινωνία με τον Ηλεκτρονικό Υπολογιστή

#	Περιγραφή	#	Περιγραφή
1	κάρτα GPS	9	Σήμα Εξωτερικού Ταλαντωτή
2	Συσκευασία	10	Ενέργεια
3	Κεραία GPS	11	Radio
4	σήμα GPS	12	Καταγραφικό
5	σήμα COM1	13	Εξωτερική πηγή DC
6	σήμα COM2	14	Συσκευή USB
7	σήμα USB	15	Εξωτερικός Ταλαντωτής
8	σήμα COM3	16	PC

Πίνακας 3.2: Υπόμνημα Σχήματος 3.3

3.4 Ξεκινώντας τη λειτουργία του δέκτη GPS

Το λογισμικό του δέκτη βρίσκεται σε μια μνήμη τύπου ROM(read-only memory). Ο δέκτης GPS ελέγχει, κατά τη διαδικασία φόρτωσης του λογισμικού του, τη σωστή λειτουργία των κυκλωμάτων του (αναλογικών - ψηφιακών). Σε περίπτωση που υπάρχει κάποιο σφάλμα εμφανίζεται ένα μήνυμα λάθους στις επικεφαλίδες των μηνυμάτων του δέκτη GPS. Όταν ο δέκτης GPS ξεκινάει για πρώτη φορά δεν μεταδίδεται καμία πληροφορία για την κατάστασή του εκτός από το όνομα της σειριακής θύρας επικοινωνίας που είναι συνδεδεμένη με την εξωτερική συσκευή ελέγχου (υπολογιστής, τερματικό). Έτσι θα δούμε στην οθόνη ένα από τα παρακάτω μηνύματα:

[COM1] Αν είναι συνδεδεμένος ο υπολογιστής στη θύρα COM1

[COM2] Αν είναι συνδεδεμένος ο υπολογιστής στη θύρα COM2

ή

[COM3] Αν είναι συνδεδεμένος ο υπολογιστής στη θύρα COM3

Όλες οι εντολές μπορούν να εισαχθούν στο πληκτρολόγιο του τερματικού μας και να εκτελεστούν από το δέκτη, αφού τις τερματίσουμε με το χαρακτήρα αλλαγής γραμμής(<Enter>).

Ένα παράδειγμα:

[COM2] FIX POSITION 51.11635 -114.0383 1048.2 [Enter]
<OK

Η επιβεβαίωση εκτέλεσης της εντολής έρχεται στην οθόνη του τερματικού με την εμφάνιση των χαρακτήρων «<OK» στην περίπτωση που είναι επιτυχής και των «<Invalid message ID» στην περίπτωση που υπάρχει κάποιο σφάλμα.

3.5 Μηνύματα δέκτη GPS

Ο δέκτης GPS χειρίζεται όλα τα εισερχόμενα και εξερχόμενα δεδομένα σε τρεις διαφορετικές μορφές μηνυμάτων: Σε συντετμημένα ASCII, ASCII και δυαδικά (binary). Αυτό αφήνει περιθώρια χρησιμοποίησης των δεκτών GPS της σειράς OEM4 σε πολλές εφαρμογές. Ο δέκτης επιπλέον υποστηρίζει μορφές μηνυμάτων του τύπου RTCA, RTCN,CMR και NMEA.
Όταν εισάγεται μία εντολή ASCII προκειμένου να ζητηθεί ένα μήνυμα από το δέκτη, ο τύπος μηνύματος που επιστρέφεται στην οθόνη του υπολογιστή καθορίζεται από τον καταληκτικό χαρακτήρα του ονόματος του μηνύματος. Το ‘A’ δείχνει ότι το μήνυμα είναι ASCII και το ‘B’ δυαδικό. Έλλειψη καταληκτικού χαρακτήρα σημαίνει πως το μήνυμα είναι συντετμημένο ASCII.

Ο Πίνακας 3.3 περιγράφει τους τύπους των πεδίων που χρησιμοποιούνται στην περιγραφή των ASCII και δυαδικών μηνυμάτων.

ΤΥΠΟΣ	Bytes	Περιγραφή
Char	1	Ο τύπος Char είναι ένας ακέραιος αριθμός 8bit με εύρος τιμών από -128 μέχρι +127.
UChar	1	Ο τύπος uchar είναι ένας ακέραιος χωρίς πρόσημο 8-bit, με εύρος τιμών από +0 μέχρι +255.
Short	2	Ακέραιος 16-bit με εύρος -32768 μέχρι +32767.
Ushort	2	16-bit χωρίς πρόσημο, με εύρος από +0 μέχρι +65535
Long	4	Ακέραιος 32-bit με εύρος από -2147483648 μέχρι +2147483647
Ulong	4	Όπως ο προηγούμενος τύπος Long αλλά δίχως πρόσημο, με εύρος από +0 μέχρι +4294967295
Double	8	64-bits: 1 bit για το πρόσημο, 11 bits για τον εκθέτη και 52 bits για το δεκαδικό μέρος.
Float	4	32-bits: 1 bit για το πρόσημο, 8 bits για τον εκθέτη και 23 bits για το δεκαδικό μέρος.
Enum	4	4-byte
GPSec	4	Η παράμετρος χρόνου για τη δυαδική έξοδο εκφράζεται σε ms και είναι τύπου long.
Hex	n	Σε δυαδική μορφή είναι πίνακας σταθερού μήκους n και σε ASCII μετατρέπεται σε δύο ζεύγη δεκαεξαδικών χαρακτήρων.
String	n	Συμβολοσειρά από bytes μεταβλητού μήκους.

Πίνακας 3.3: Τύποι πεδίων μηνύματος δέκτη.

3.5.1 Μηνύματα ASCII

Τα μηνύματα ASCII είναι αναγνώσιμα από το χρήστη και από τον υπολογιστή. Οι δομές όλων των μηνυμάτων ASCII ακολουθούν τις εξής γενικές συμβάσεις:

1. Ο αρχικός κωδικός αναγνώρισης για κάθε εγγραφή ενός μηνύματος ASCII είναι το σύμβολο ‘#’
2. Κάθε μήνυμα ή εντολή είναι μεταβλητού μήκους ανάλογα με το μέγεθος των δεδομένων και των μορφοποιήσεων.
3. Όλα τα πεδία των δεδομένων χωρίζονται με κόμμα (comma delimited).
4. Κάθε μήνυμα λήγει σε ένα δεκαεξαδικό αριθμό του οποίου προηγείται ένας αστερίσκος και ακολουθεί [CR][LF]. Αυτή η τιμή είναι 32 bit CRC για όλα τα bytes του μηνύματος, συμπεριλαμβανομένου του ‘#’ και του ‘*’.
5. Μια συμβολοσειρά ASCII είναι ένα πεδίο εντός εισαγωγικών
   Παράδειγμα:

#BESTXYZA,COM1,0,72.5,FINESTEERING,1058,404080.000,00000028,06e1,37;SOL_COMPUTED,SINGLE,-1634502.769,3664601.068,4942471.986,26.3812,34.7726,38.3204,SOL_COMPUTED,DOPPLER_VELOCITY,0.195,0.496,-0.254,3.1532,4.1561,4.5802,**"AAAA"**,0.250,0.000,0.000,6,6,0,0,0,0,0,0*958fe88c

6. Αν ο δέκτης εντοπίσει ένα λάθος κατά τη συντακτική ανάλυση της εντολής θα επιστρέψει μήνυμα λάθους.

Το μήνυμα ASCII δομείται όπως περιγράφεται στον Πίνακα 3.4:

Πεδίο #	Όνομα Πεδίου	Τύπος Πεδίου	Περιγραφή
1	Sync	Char	Χαρακτήρας συγχρονισμού.
2	Message	Char	Το όνομα ASCII του μηνύματος.
3	Port	Char	Το όνομα της θύρας από όπου το μήνυμα δημιουργήθηκε.
4	Sequence #	Long	Χρησιμοποιείται για σχετικά μηνύματα. Συνήθως είναι 0.
5	Idle Time	Float	Το ελάχιστο ποσοστό του χρόνου που ο επεξεργαστής είναι αδρανής.
6	GPS Time Status	Enum	Η ποιότητα του χρόνου GPS
7	Week	Ulong	Ο αριθμός της εβδομάδας του χρόνου του GPS.
8	Seconds	GPSec	Δευτερόλεπτα από την αρχή της εβδομάδας GPS.
9	Receiver Status	Ulong	Οκταψήφιος δεκαεξαδικός αριθμός που αναπαριστά την κατάσταση του δέκτη.
10	Reserved	Ulong	Κρατείται για εσωτερική χρήση.
11	Receiver s/w Version	Ulong	Τιμή (0 - 65535) που δείχνει τον αριθμό έκδοσης του λογισμικού.
12	;	Char	Τέλος επικεφαλίδας.

Πίνακας 3.4: Επικεφαλίδα ASCII μηνύματος.

3.5.2 Συντετμημένα Μηνύματα ASCII

Αυτή η μορφοποίηση μηνύματος έχει σχεδιαστεί για να καταστήσει την εισαγωγή και την προβολή των μηνυμάτων όσο πιο απλή γίνεται για το χρήστη. Τα δεδομένα αναπαρίστανται ως απλοί χαρακτήρες ASCII, χωρισμένοι με κενούς χαρακτήρες ή κόμματα και ταξινομημένοι με τέτοιο τρόπο, ώστε να είναι εύληπτοι.

Παράδειγμα εντολής:

log com1 loglist

Μήνυμα επιστροφής:

<LOGLIST COM1 0 69.0 FINE 0 0.000 00240000 206d 0
< 4
< COM1 RXSTATUSEVENTA ONNEW 0.000000 0.000000 NOHOLD
< COM2 RXSTATUSEVENTA ONNEW 0.000000 0.000000 NOHOLD
< COM3 RXSTATUSEVENTA ONNEW 0.000000 0.000000 NOHOLD
< COM1 LOGLIST ONCE 0.000000 0.000000 NOHOLD

3.5.3 Δυαδικά Μηνύματα

Αυτά τα μηνύματα είναι αυστηρώς αναγνώσιμα μόνο από μηχανές. Επίσης είναι ιδανικά για εφαρμογές όπου μεταφέρεται μεγάλος όγκος δεδομένων. Εξαιτίας της εγγενούς συμπύκνωσης των δυαδικών σε σχέση με τα ASCII, τα μηνύματα είναι πολύ μικρότερα σε μέγεθος (bytes). Αυτό επιτρέπει σε ένα μεγαλύτερο μέγεθος δεδομένων να μεταδοθούν και να παραληφθούν από τις σειριακές θύρες επικοινωνίας του δέκτη. Η δομή όλων των δυαδικών μηνυμάτων ακολουθεί τις εξής γενικές συνθήκες:

1. Βασική μορφοποίηση:

Επικεφαλίδα	3 bytes συγχρονισμού + 25 bytes της πληροφορίας της επικεφαλίδας.
Δεδομένα	Μεταβλητά
CRC	4 bytes

2. Τα 3 bytes συγχρονισμού είναι πάντα:

Byte	Hex	Decimal
Πρώτο	AA	170

Δεύτερο	44	68
Τρίτο	12	18

Πίνακας 3.5: 3 byte ελέγχου μηνύματος ASCII

3. Το CRC είναι 32 bit CRC
4. Η επικεφαλίδα είναι σε μορφοποίηση που φαίνεται στον ακόλουθο πίνακα (3.6)

Πεδίο #	Όνομα Πεδίου	Τύπος Πεδίου	Περιγραφή	Bytes	Offset
1	Sync	Char	Δεκαεξαδικός 0xAA	1	0
2	Sync	Char	Δεκαεξαδικός 0x44	1	1
3	Sync	Char	Δεκαεξαδικός 0x12	1	2
4	Header Length	Uchar	Μέγεθος της επικεφαλίδας	1	3
5	Message ID	Ushort	Ο αριθμός ID από το μήνυμα	2	4
6	Message Type	Char	Bits 0-4 = Reserved Bits 5-6 = Format 00 = Binary 01 = ASCII 10 = Abbreviated ASCII, NMEA 11 = Reserved Bit 7 = Response 0 = Original Message 1 = Response Message	1	6
7	Port Address	Char	Ο αριθμός της εβδομάδας του χρόνου του GPS.	1	7
8	Message Length	Ushort	Το μέγεθος του σώματος του μηνύματος χωρίς το CRC.	2	8
9	Sequence	Ushort	Χρησιμοποιείται για σχετικά μηνύματα. Συνήθως είναι 0.	2	10
10	Idle Time	Char	Το ελάχιστο ποσοστό του χρόνου που ο επεξεργαστής είναι αδρανής.	1	12
11	Time Status	Enum	Η ποιότητα του χρόνου GPS	1[c]	13
12	Week	Ushort	Ο αριθμός της εβδομάδας του χρόνου του GPS.	2	14
13	Milliseconds	GPSec	milliseconds από την αρχή της εβδομάδας GPS	4	16
14	Receiver Status	Ulong	32 bit αριθμός που αναπαριστά την κατάσταση του δέκτη	4	20
15	Reserved	Ushort	Κρατείται για εσωτερική χρήση	2	24
16	Receiver s/w version	Ushort	Τιμή (0 - 65535) που δείχνει τον αριθμό έκδοσης του λογισμικού	2	26

Πίνακας 3.6: Μορφοποίηση επικεφαλίδας μηνύματος

3.5.4 Αποκρίσεις δέκτη

Κανονικά, με την εισαγωγή μιας εντολής από το τερματικό στο δέκτη του GPS, αυτός επιστρέφει πίσω στο τερματικό ένα μήνυμα απόκρισης. Αν είναι επιθυμητό η εντολή INTERFACEMODE μπορεί να χρησιμοποιηθεί για να απενεργοποιήσει τα μηνύματα απόκρισης. Η απόκριση του δέκτη θα έχει την ίδια μορφοποίηση με εκείνη της εντολής που εισήχθη (δυαδικό εισαγόμενο = δυαδική απάντηση).

Απάντηση ASCII:

Αποτελείται από την επικεφαλίδα του μηνύματος, μόνο που στο τέλος του ονόματος προστίθεται το 'R' (Response) όπως φαίνεται και στο παράδειγμα

Π.χ:

#BESTPOS**R**,COM1,0,67.0,FINE,1028,422060.400,00000000,a31b,0;"OK"
*b867caad

Δυαδική απάντηση:

Παρόμοια με μια ASCII απάντηση με τη διαφορά ότι ότι ακολουθεί δυαδικό πρωτοκολλο.

3.5.5 Χρονική σφραγίδα μηνύματος

Όλα τα μηνύματα NovAtel που παράγονται από τους OEM4 δέκτες έχουν μια χρονική σφραγίδα στην επικεφαλίδα τους. Ο χρόνος GPS αντιστοιχεί σε UTC χρόνο με το σημείο αναφοράς (μηδέν) να ορίζεται τα μεσάνυχτα της $5^{ης}$ Ιανουαρίου 1980. Η χρονική σφραγίδα αποτελείται από τον αριθμό εβδομάδων GPS (GPS week) από το σημείο μηδέν και τον αριθμό δευτερολέπτων που πέρασαν από την τελευταία εβδομάδα (0 έως 603.799). Ο χρόνος GPS διαφέρει από το χρόνο UTC δεδομένου ότι τα δευτερόλεπτα διόρθωσης του δέκτη NovAtel (leap seconds) παρεμβάλλονται περιστασιακά στο χρόνο UTC. Επιπλέον, ένα μικρό λάθος (λιγότερο από 1ms) μπορεί να υπάρξει στο συγχρονισμό μεταξύ του χρόνου UTC και του χρόνου αναφοράς του

GPS. Το μήνυμα TIME αναφέρει το χρόνο GPS και UTC και τη διαφορά μεταξύ των δύο.

Τα δεδομένα στα σύγχρονα μηνύματα (π.χ. RANGE, BESTPOS, TIME) είναι βασισμένα σε μια περιοδική μέτρηση της ψευδοαπόστασης του δορυφόρου. Η χρονική σφραγίδα σε αυτά τα μηνύματα προσδιορίζεται από το δέκτη, σε χρόνο GPS, τη στιγμή που γίνεται η μέτρηση. Ένα σύγχρονο μήνυμα που λαμβάνεται σε ONTIME 1 μπορεί να χρησιμοποιηθεί από κοινού με το 1PPS σήμα για να παρέχει σχετική ακρίβεια καλύτερη από 250 ns.

Άλλοι τύποι μηνυμάτων (ασύγχρονοι) προκαλούνται από ένα εξωτερικό γεγονός και ο χρόνος στην επικεφαλίδα τους τυχόν να μη μπορεί να συγχρονιστεί στον τρέχοντα χρόνο GPS. Μηνύματα που περιέχουν τα δορυφορικά στοιχεία ράδιο-μετάδοσης (π.χ. ALMANAC, GPSEPHEM) έχουν το χρόνο μετάδοσης του τελευταίου subframe τους στην επικεφαλίδα. Εντούτοις, όταν τα ασύγχρονα μηνύματα προκαλούνται ONTIME, η χρονική σφραγίδα αντιπροσωπεύει το χρόνο που το μήνυμα παρήχθη και όχι το χρόνο που δόθηκε από τα δεδομένα του.

Η υλοποίηση της παραπάνω διαδικασίας καθώς και των άλλων λειτουργιών του λογισμικού NORD θα περιγραφούν αναλυτικά στο παρακάτω Κεφάλαιο.

Κεφάλαιο 4

Λογισμικό NORD (NovAtel Oem4 Receiver Daemon)

4.1 Εισαγωγή

Το λογισμικό NORD ελέγχου του δέκτη NovAtel αναπτύχθηκε πάνω σε μια αρχιτεκτονική πολλαπλών επιπέδων (n-tier) ώστε να εξασφαλίζεται η αυτονομία ανάπτυξης των επιπέδων υλοποίησης. Έτσι για οποιαδήποτε αλλαγή στον κώδικα δεν χρειάζεται να ξαναγραφτεί η εφαρμογή από την αρχή αλλά μόνο να αλλαχθεί το συγκεκριμένο επίπεδο εντολής (layer). Το λογισμικό χωρίζεται σε δύο μέρη: σε αυτό του διακομιστή (server) και σε αυτό του πελάτη (client). Ο πελάτης είναι ο αιτών των υπηρεσιών και ο διακομιστής απαντάει στις αιτήσεις που γίνονται από τους "πελάτες" (client). Έτσι επιτυγχάνεται η επικοινωνία με τον δέκτη και ο έλεγχος του από εξωτερικούς χρήστες. Παρακάτω θα περιγραφεί η μεθοδολογία ανάπτυξης του λογισμικού NORD.

4.2 Περιγραφή αρχιτεκτονικής

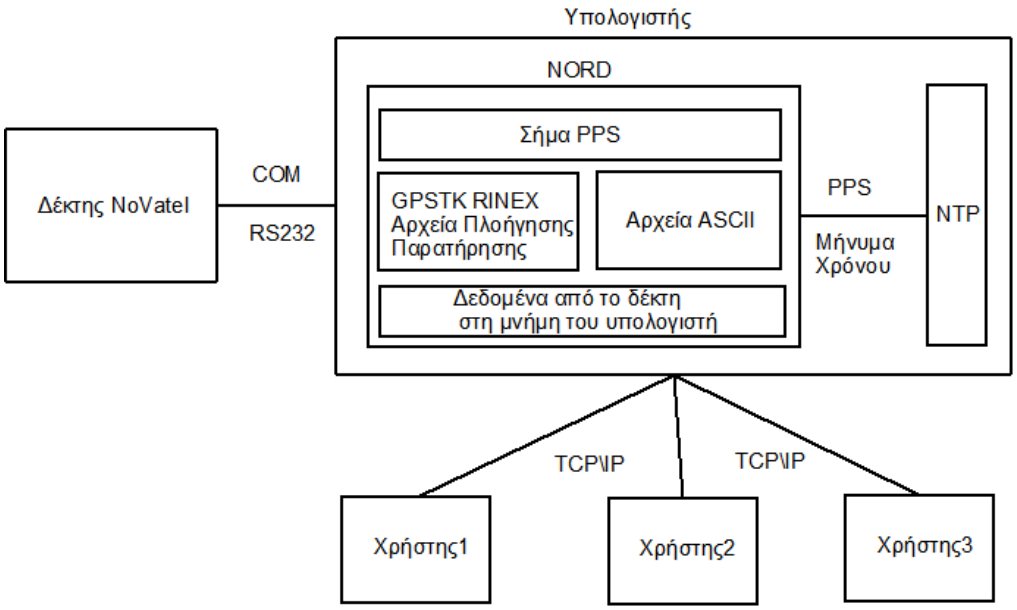
Το λογισμικό NORD αρχικά υλοποιεί ένα TCP/IP κόμβο επικοινωνίας για να δέχεται τις συνδέσεις από τους πελάτες. Ανάλογα με το πόσοι χρήστες από ένα ή περισσότερα τερματικά θέλουν να συνδεθούν στον δέκτη NovAtel τόσες είναι και οι αφιερωμένες και αυτόνομες συνδέσεις που ανοίγονται για τον κάθε χρήστη. Έπειτα ο χρήστης πληκτρολογεί και αποστέλλει τις εντολές που θέλει να εκτελέσει ο δέκτης. Στην οθόνη του τερματικού του μπορεί να δει αν η εντολή εκτελέστηκε με επιτυχία ή όχι, τα τυχόν λάθη που έκανε ίδιος κατά την πληκτρολόγηση, αλλά και την αδυναμία του δέκτη να την εκτελέσει. Ο χρήστης επικοινωνεί με τον δέκτη, είτε μέσα από μια ειδικά διαμορφωμένη γραμμή εντολών, είτε από γραφικό περιβάλλον που παρέχεται από έναν web browser.

Το λογισμικό NORD διαβάζει δεδομένα που προέρχονται από το δέκτη μέσω της σειριακής θύρας επικοινωνίας του υπολογιστή COM1, COM2 που είναι συνδεδεμένη

με την αντίστοιχη σειριακή θύρα του δέκτη. Τα δεδομένα αυτά αποθηκεύονται στη μνήμη του υπολογιστή. Πιο συγκεκριμένα, το λογισμικό λαμβάνει συνεχώς δεδομένα από το δέκτη και προσπαθεί να αναγνωρίσει κάποιο μήνυμά του. Όταν συμβεί αυτό, το μήνυμα μπαίνει σε μια σειρά αναμονής.

Συγχρόνως ξεκινά μια άλλη αυτόνομη διαδικασία η οποία αφαιρεί ένα ένα τα μηνύματα από την ουρά. Καθώς τα αφαιρεί ελέγχει το είδος του μηνύματος και αν αυτό είναι RANGE ή RAWEPHEM γράφει στο δίσκο του υπολογιστή και τα κατάλληλα αρχεία RINEX σε θέσεις που έχουν υποδειχτεί προηγουμένως από τον χρήστη. Ουσιαστικά το λογισμικό NORD υλοποιεί ένα μοντέλο παραγωγού-καταναλωτή στο οποίο ο πρώτος παράγει μηνύματα του δέκτη και ο δεύτερος τα καταναλώνει γράφοντας τα σε κάποιο αποθηκευτικό μέσο ή τα επιστρέφει στον χρήστη σε μια πιο αναγνώσιμη μορφή.

Παράλληλα μια τρίτη αυτόνομη διαδικασία παρακολουθεί την αλλαγή κατάστασης από 0 σε 1 που συμβαίνει στο pin 8 CTS (Clear to Send) της σειριακής θύρας, η οποία συμβαίνει κάθε 1 δευτερόλεπτο ακριβώς. Το σήμα 1PPS (Pulse Per Second), όπως ονομάζεται, προκαλείται από το δέκτη NovAtel και υπάρχει η δυνατότητα να ενεργοποιηθεί ή να απενεργοποιηθεί. Έπειτα το λογισμικό αναλαμβάνει να ενημερώσει έναν time-server μέσω μια κοινής μνήμης που διαθέτουν. Με αυτό τον τρόπο παρέχεται στο διαδίκτυο ένας χρόνος ακρίβειας ενός δευτερολέπτου.



Σχήμα 4.1: Αρχιτεκτονική NORD

4.3 Ανάπτυξη λογικής και κώδικα του λογισμικού NORD

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του λογισμικού είναι η C++ και οι βιβλιοθήκες της. Η γλώσσα C++ είναι από το 1990, μια από τις πιο διάσημες γλώσσες προγραμματισμού και ουσιαστικά η συνέχεια της γλώσσας C με πολλά επιπλέον χαρακτηριστικά. Μερικά από αυτά όπως ο αντικειμενοστραφής (object-oriented) προγραμματισμός, οι αφηρημένοι και γενικοί τύποι δεδομένων (Templates, Generics) είναι και ο λόγος που η ανάπτυξη του λογισμικού NORD έγινε πάνω σ' αυτή. Η μεταγλώττιση (compilation) του λογισμικού έγινε με τον μεταγλωττιστή gcc στη πλατφόρμα του λειτουργικού συστήματος Debian Linux. Παρακάτω θα περιγραφούν αναλυτικά οι ρουτίνες και οι μέθοδοι που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.

4.3.1 Κλάση ρυθμίσεων (Configuration Class)

Το αρχείο ρυθμίσεων (Configuration file) του NORD βρίσκεται στην τοποθεσία "/etc/nord/nord.conf" και περιέχει πληροφορίες για τη σειριακή θύρα επικοινωνίας, τα στοιχεία σύνδεσης και τα αρχεία RINEX. Ένα παράδειγμα του αρχείου ρυθμίσεων φαίνεται παρακάτω:

```
#Serial Port Settings
port = /dev/ttyS0
baudrate = B19200
#Server Listens
host=localhost
port=6000
#Rinex Files Settings
Obsinterva l= 15
Navinterval = 60
NavFilesPath = /home/gpapadak/diplomatique/prog/c++/Navs/
ObsFilesPath = /home/gpapadak/diplomatique/prog/c++/Obss/
FileProgram = NovaR v1.0
Observer = George Papadakis
Agency = TUC
```

Το λογισμικό NORD ουσιαστικά αναλαμβάνει την ανάγνωση και εγγραφή των ρυθμίσεων στο σκληρό δίσκο του υπολογιστή.

4.3.2 Διαχείρηση σειριακής θύρας επικοινωνίας

Το λογισμικό NORD ανοίγει τη σειριακή θύρα επικοινωνίας με την ακόλουθη μέθοδο της C++.

```
int fn;
/* File descriptor for the port */
//      fd = open(MODEMDEVICE, O_RDWR );
fn = open (port.c_str(), O_RDWR | O_NOCTTY | O_NONBLOCK);
```

Σε αυτό το σημείο παρατηρήθηκε το εξής πρόβλημα με το δέκτη. Κάθε φορά που εκτελούνταν η παραπάνω μέθοδος και δοκιμάζαμε να γράψουμε ή να διαβάσουμε τη σειριακή θύρα επικοινωνίας, ο δέκτης έκανε επανεκκίνηση (Reset) με αποτέλεσμα να πρέπει να περάσουν περίπου 15 δευτερόλεπτα μέχρι να είναι έτοιμος για εκ νέου λειτουργία. Αυτό το γεγονός προκαλούσε μια δυσλειτουργία στη διαδικασία. Το πρόβλημα αντιμετωπίστηκε με τη προσθήκη του παρακάτω κώδικα:

```
ioctl (fn, TIOCMGET, &status);
status &= ~TIOCM_RTS;
ioctl (fn, TIOCMSET, &status);
```

Αλλάζουμε δηλαδή την κατάσταση στο pin 4 DTR (Data Terminal Ready) της σειριακής θύρας επικοινωνίας και το τοποθετούμε στο 0. Έτσι ο δείκτης αρχείου fn (file descriptor) είναι έτοιμος για εγγραφή και ανάγνωση.

4.3.3 Δημιουργία αυτόνομων γραμμών επεξεργασίας (threads)

Συνολικά δημιουργούνται τρεις αυτόνομες γραμμές επεξεργασίας. Μια για τον έλεγχο των παραγόμενων χρονικών σημάτων 1sec του PPS και δύο για τη διαχείριση των μηνυμάτων του δέκτη. Τα thread1 και thread2 που έχουν τους «ρόλους παραγωγού – καταναλωτή» (αλγόριθμος producer – consumer) αντίστοιχα ρυθμίζονται από ένα κλειδί (mutex). Ακριβώς επειδή και τα δύο απασχολούν τους

ίδιους πόρους μνήμης, την ουρά αναμονής ουσιαστικά, κλειδώνουν και ξεκλειδώνουν εναλλάξ το mutex αποφεύγοντας έτσι την επικάλυψη στα δεδομένων. Το thread3 όπως προαναφέραμε παρακολουθεί την αλλαγή κατάστασης από 0 σε 1 που συμβαίνει στο pin 8 CTS (Clear to Send) της σειριακής πόρτας του υπολογιστή που τρέχει το NORD. Έπειτα το κύριο νήμα του προγράμματος δηλαδή η main() μέθοδος περιμένει για συνδέσεις της θύρας δικτύου στο Socket που μόλις άνοιξε.

4.3.4 Δημιουργία Θύρας δικτύου (Socket)

Αφού ξεκινήσουν την εκτέλεση τους όλα τα νήματα το κύριο νήμα πάνω στο οποίο εκτελείται η main() ανοίγει μια θύρα δικτύου TCP/IP socket στην διεύθυνση IP και θύρα, τα οποία δώσαμε αρχικά μέσα από το αρχείο ρυθμίσεων (configuration file). Όταν ένας χρήστης ανοίγει μια σύνδεση με το λογισμικό NORD, αυτόματα δημιουργείται μια καινούργια σύνδεση στη θύρα δικτύου και παραμένει ανοικτή μέχρι να την τερματίσει ο χρήστης. Τη λογική λειτουργίας της θύρας δικτύου και των συνδέσεων σε αυτό υλοποιεί η κλάση ServerSocket. Κατά τη διάρκεια αυτής της σύνδεσης ο χρήστης αποστέλλει και δέχεται μηνύματα από και προς τον δέκτη. Κατά την αποστολή εντολής, σε μορφή συμβολοσειράς, προς το δέκτη, το socket αποθηκεύει τη συμβολοσειρά προσωρινά σε μια μεταβλητή η οποία ελέγχεται αργότερα από τη μέθοδο CmdParse(). Η μέθοδος CmdParse() αναλαμβάνει να ελέγξει αν η εντολή αλλάζει κάποια ρύθμιση στη λειτουργία του NORD ή αποτελεί αίτημα του χρήστη για κάποιο μήνυμα από το δέκτη. Στη δεύτερη περίπτωση, η συμβολοσειρά γράφεται στη σειριακή θύρα επικοινωνίας του δέκτη κατάλληλα μορφοποιημένη. Αφού ολοκληρωθεί η επεξεργασία της εντολής, αποστέλλεται πίσω στο χρήστη μέσα από την ίδια σύνδεση ένα μήνυμα ενημέρωσης για το αποτέλεσμα.

4.4 Επεξεργασία μηνύματος δέκτη

Όλα τα μηνύματα του δέκτη NovAtel αναγνωρίζονται από το λογισμικό NORD με τη βοήθεια της κλάσης GpsOutC βάσει της πληροφορίας που φέρουν στις επικεφαλίδες τους. Συγκεκριμένα το Message ID, πεδίο της επικεφαλίδας, μας δίνει τον αριθμό του μηνύματος του δέκτη. Όλα τα μηνύματα που αναγνωρίζονται αποθηκεύονται σε ετερογενή ουρά εντολών. Η ουρά ονομάζεται ετερογενής γιατί το κάθε μήνυμα

διαφέρει από τα όλα ως προς τον τύπο των δεδομένων του που περιέχει. Έτσι όλα τα μηνύματα μπαίνουν στην ουρά αναμονής κατά τη σειρά που διαβάστηκαν από το λογισμικό. Ενδιαφέρον παρουσιάζουν τα μηνύματα που περιέχουν δεδομένα παρατήρησης (observation data) και ναυσιπλοΐας (navigation data) τα οποία είναι αυτά που γράφονται σε αρχεία RINEX στο δίσκο του υπολογιστή σε πραγματικό χρόνο. Οι εντολές που απαιτούνται να δοθούν στο δέκτη είναι οι «log rangeb ontime 1» και «log rawephemb onchanged» αντίστοιχα και τα δεδομένα που περιέχουν παρουσιάζονται στους Πίνακες 4.1 και 4.2.

Field	Field type	Data Description	Format	Binary Bites	Binary Offset
1	Header	Log header		H	0
2	# obs	Number of observations with information to follow[a]	Long	4	H
3	PRN	GPS satellite PRN number of range measurement	UShort	2	H+4
4	Reserved		UShort	2	H+6
5	psr	Pseudorange measurement (m)	Double	8	H+8
6	psr std	Pseudorange measurement standar deviation (m)	Float	4	H+16
7	adr	Carrier phase, in cycles (accumulated Doppler range)	Double	8	H+20
8	adr std	Estimated carrier phase standar deviation (cycles)	Float	4	H+28
9	dopp	Instantaneus carrier Doppler frequency (Hz)	Float	4	H+32
10	C/N_0	Carrier to noise density ratio $C/N_0 = 10[\log_{10}(S/N_0)]$(dB-Hz)	Float	4	H+36
11	locktime	Number of seconds of continuous tracking (no cycle slipping)	Float	4	H+40
12	Ch-tr-status	Tracking status	ULong	4	H+44
13	Next PRN offset = H + 4 + (#obs x 44)				
Variable	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+4 (#obs x 44)
Variable	[CR][LF]	Sentence terminator (ASCII only)	-	-	-

Πίνακας 4.1: Εντολή Range Log.

Field	Field type	Data Description	Format	Binary Bytes	Binary Offset
1	header	Log header		H	0
2	prn	Satellite PRN number	Ulong	4	H
3	ref week	Ephemeris reference week number	Ulong	4	H+4
4	ref secs	Ephemeris reference time (seconds)	Ulong	4	H+8
5	subframe 1	Subframe 1 data.	Hex	30	H+12
6	subframe 2	Subframe 2 data.	Hex	30	H+42
7	subframe 3	Subframe 3 data.	Hex	30	H+72
8	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+102
9	[CR][LF]	Sentence terminator (ASCII only)	-	-	-

Πίνακας 4.2: Εντολή Rawephem Log.

Όλα τα υπόλοιπα μηνύματα του δέκτη NovAtel αποθηκεύονται σε ένα αρχείο ASCII που ορίζει ο χρήστης ή ανακατευθύνονται στην οθόνη του χρήστη μέσω της θύρας δικτύου TCP\IP socket και της σύνδεσης μετρήσεων GPS που έχει δημιουργηθεί.

4.4.1 Δεδομένα τύπου Rinex

Τα δεδομένα τύπου RINEX (Receiver Independent Exchange Format) είναι του τύπου ASCII και χρησιμοποιείται διεθνώς με σκοπό τη δυνατότητα επεξεργασίας δεδομένων από δέκτες διαφορετικών εταιριών με οποιοδήποτε λογισμικό GPS. Το NORD έχει τη δυνατότητα μετατροπής των δεδομένων OEM4 GPS, παρατηρήσεων (observation) και ναυσιπλοΐας (navigation), σε αρχεία παρατηρήσεων με RINEX format. Αυτό επιτυγχάνεται με τη χρήση της πλατφόρμας GPSTk και των βιβλιοθηκών της. Η πλατφόρμα GPSTk (δημιουργήθηκε από ομάδα του πανεπιστημίου του TEXAS των ΗΠΑ) είναι ένα σύνολο με θεμελιώδεις και εξειδικευμένους αλγορίθμους για την επεξεργασία των σημάτων GPS. Πρόκειται για ένα λογισμικό ανοιχτού κώδικα που προσφέρει μεταξύ άλλων, βιβλιοθήκες για την επικοινωνία σε format RINEX, για τον υπολογισμό δορυφορικών εφημερίδων, για ατμοσφαιρικά μοντέλα διάθλασης καθώς και αλγορίθμους τοποθέτησης.

4.5 Δημιουργία ενός παλμού ανά δευτερόλεπτο 1PPS (1 Pulse Per Second) Σήμα

Ο δέκτης NovAtel έχει τη δυνατότητα να μεταβάλει τη κατάσταση του RTS pin της σειριακής θύρας επικοινωνίας, που αντιστοιχεί στο CTS pin της σειριακής θύρας επικοινωνίας του υπολογιστή που εκτελείται το NORD, από 0 σε 1 παλμό κάθε ένα δευτερόλεπτο. Αυτό γίνεται δίνοντας την εξής εντολή στο δέκτη:

COMCONTROL COM3 RTS TOGGLEPPS

Γι' αυτό το λόγο αναπτύχθηκε μια διαδικασία (η κλάση *ntpshm*) η οποία δίνει πρόσβαση στη shared memory-segment του ntpd. Παρακάτω φαίνεται η δομή του τμήματος μνήμης (memory segment):

```
#define NTPD_BASE   0x4e545030      /* "NTP0" */
#define SHM_UNIT    0               /* SHM driver unit number (0..3) */
#define NTPSHMSEGS 4                /* number of NTP SHM segments */

typedef struct  {
   int    mode; /* 0 - if valid set
                  *       use values,
                  *       clear valid
                  * 1 - if valid set
                  *         if count before and after read of values is equal,
                  *       use values
                  *       clear valid
                  */
   int    count;
   time_t clockTimeStampSec;
   int    clockTimeStampUSec;
   time_t receiveTimeStampSec;
   long   receiveTimeStampUSec;
   int    leap;
   int    precision;
   int    nsamples;
   int    valid;
   int    pad[10];
}shmTime;
```

Όπου NTPD_BASE 0x4e545030 είναι η θέση μνήμης του ntp segment.

Ο ntpd είναι ένα λογισμικό το οποίο διατηρεί και συγχρονίζει την ώρα ενός συστήματος βάση άλλων χρονομέτρων τα οποία συνήθως είναι προσβάσιμα από το διαδίκτυο. Είναι μια υλοποίηση του Network Time Protocol (NTP, version 4). Για να έχει ο ntpd ως χρονόμετρο αναφοράς το σήμα PPS του δέκτη και την ώρα που παίρνουμε από το δέκτη μέσω της αποκωδικοποίησης του μηνύματος TIME, ακολουθήθηκε η εξής διαδικασία. Όπως είπαμε το thread3 ελέγχει αν η κάθε αλλαγή της κατάστασης από 0 σε 1 και αντίστροφα διαρκεί ένα δευτερόλεπτο. Στην περίπτωση που αυτό συμβαίνει, εγγράφουμε τον τωρινό χρόνο στο τμήμα μνήμης του ntpd με τον ακόλουθο τρόπο:

```
#define timediff(x, y)     (int)((x.tv_sec-y.tv_sec)*1000000+x.tv_usec-y.tv_usec)
              cycle = timediff(tv, pulse[state]);
              duration = timediff(tv, pulse[state == 0]);
#undef timediff
           cout<<"PPS cycle: "<<cycle<<", duration: "<<duration<<"\n";
              if ( 800000 > duration) {
                    /* less then 800mS, duration too short for anything */
             cout<<"PPS pulse rejected too short. cycle: "<<cycle<<", duration:
"<<duration<<"\n";
              } else if (cycle > 999000 && cycle < 1001000 ) {
             /* looks like PPS pulse */
                    (void)oNtp.ntpshm_pps(&tv);
              } else if (cycle > 1999000 && cycle < 2001000) {
                    /* looks like 2Hz square wave */
                    (void)oNtp.ntpshm_pps(&tv);
           } else {
              cout<<"PPS pulse rejected.  cycle: "<<cycle<<", duration: "<<duration<<"\n";

              }
```

Δηλαδή όταν η μεταβλητή cycle (μετράει τον απαιτούμενο χρόνο σε ms, για την μετάβαση από τον 0 στο 1) παίρνει τιμές από 1999000 μέχρι 2001000, περίπου 2 δευτερόλεπτα, τότε είμαστε μέσα στο απαιτούμενο χρονικό διάστημα για έναν πλήρη κύκλο(μετάβαση από το 0 στο 1 και πάλι στο 0). Η μέθοδος που εγγράφει τον χρόνο στο τμήμα μνήμης του ntpd είναι η ntpshm_pps(struct timeval *tv). Στην περίπτωση που υπάρξει δυσλειτουργία (π.χ απώλεια ενός κύκλου), το μήνυμα TIME log του δέκτη NovAtel έρχεται να διορθώσει την ώρα. Το χρονικό μήνυμα TIME log δίνει την ακριβή ώρα UTC. Ο χρόνος στην κλίμακα UTC μετατρέπεται σε τοπικό χρόνο και εγγράφεται στο τμήμα μνήμης του ntpd με τον ακόλουθο τρόπο.

```
int NtpShm::ntpshm_put(double fixtime)
```

```
/* put a received fix time into shared memory for NTP */
{
   shmTime *myshmTime = NULL;
   struct timeval tv;
   double seconds,microseconds;

   if (shmindex < 0 ||
         (myshmTime = oshmTime[shmindex]) == NULL)
         return 0;

   (void)gettimeofday(&tv,NULL);
   microseconds = 1000000.0 * modf(fixtime,&seconds);

   myshmTime->count++;
   myshmTime->clockTimeStampSec = (time_t)seconds;
   myshmTime->clockTimeStampUSec = (int)microseconds;
   myshmTime->receiveTimeStampSec = (time_t)tv.tv_sec;
   myshmTime->receiveTimeStampUSec = tv.tv_usec;
   myshmTime->count++;
   myshmTime->valid = 1;

   return 1;
}
```

Έτσι από τη μια έχουμε ένα τέλειο χρονόμετρο NovAtel με ακρίβεια δευτερολέπτου και από την άλλη μια ακριβή ώρα της κλίμακας UTC που διορθώνει την ώρα του υπολογιστή όποτε το 1PPS σήμα αδυνατεί να τη διατηρήσει σταθερή.

ΚΕΦΑΛΑΙΟ 5

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ

Ανακεφαλαιώνοντας μπορεί να διατυπωθεί ο ισχυρισμός πως η διπλωματική εργασία που παρουσιάζεται εκπλήρωσε τον αρχικό της σκόπο. Η ανάπτυξη του λογισμικού για τη σειρά OEM4 των καρτών GPS NovAtel ολοκληρώθηκε με επιτυχία και έτσι οι κάρτες αυτές μπορούν πλέον να χρησιμοποιηθούν σε αντίστοιχες συστοιχίες επιστημονικών οργάνων χωρίς περιορισμούς. Κατά τη διάρκεια της ανάπτυξης του λογισμικού η συμμετοχή σε δικτυακούς τόπους συζητήσεων (forum) για την εύρεση βέλτιστων αλγορίθμων και μεθόδων, απέδειξε πως υπήρχε άμεση ανάγκη από την πλευρά της επιστημονικής κοινότητας για τη λειτουργία των Καρτων GPS NovAtel της σειράς OEM4. Πλήθος ερευνητών και μηχανικών από διάφορους οργανισμούς κυρίως του εξωτερικού (π.χ. από τη NASA) έδειξαν έντονο ενδιαφέρον για την πορεία της εργασίας και τα τελικά αποτελέσματα της εφαρμογής του λογισμικού σε πραγματικές συνθήκες.

Η σχετικά απλή διεπαφή ελέγχου του δέκτη με τερματικό που κατασκευάστηκε, αν και καλύπτει όλες τις λειτουργίες και τις δυνατότητες των καρτών, επιδέχεται κάποιες βελτιώσεις όσον αφορά την λειτουργικότητα της (όπως π.χ η μορφοποίηση του συνόλου των απαντήσεων του δέκτη GPS, καθώς και η αποθήκευση των μετεωρολογικων δεδομένων που παρέχονται από αυτόν). Αν στο μέλλον γίνει ευρύτερη χρήση του λογισμικού μπορεί να παρουσιαστεί η ανάγκη για μια τέτοιου είδους «επιδιορθωτική» παρέμβαση στο λογισμικό, γεγονός που έχει προβλέφθεί μέσα στον κώδικα και μπορεί να ολοκληρωθεί χωρίς μεγάλες δυσκολίες.

Στα πλάισια της ανοικτής επιστημονικής συνεργάσιας σε παγκόσμιο επίπεδο προβλέπεται η κατασκεύη ενός ιστότοπου όπου ενδιαφερόμενοι θα μπορούν να ενημερώνονται, για τις μεθόδους που χρησιμοποιήθηκαν στο λογισμικό, για την προσαρμογή του λογισμικού σε άλλες κάρτες GPS δεκτών, για πληροφορίες σχετικές με την αναβάθμιση του καθώς και να προτείνουν βελτιώσεις ή να επισημάνουν τυχόν δυσλειτουργίες. Τα παραπάνω θα καταστούν αναγκαία όταν γίνει ευρύτερη χρήση του λογισμικού από διάφορες επιστημονικές ομάδες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Μερτίκας Σ. Π. (1999). Τηλεπισκόπηση και Ψηφιακή Ανάλυση Εικόνας, Εκδοτικός Όμιλος Ίων, Αθήνα.
- Μπάρτζος Η. Δ., (1998), *Μελέτη και Αξιολόγηση των Επιδράσεων της Ιονόσφαιρας στον Σταθμό Αναφοράς GPS του Πολυτεχνείου Κρήτης,* Μεταπτυχιακή Διατριβή, Χανιά.
- NovAtel (2004). *OEM4 Family of Receivers Installation & Operation Manual Volume 1*, Canada.
- NovAtel (2004). *OEM4 Family of Receivers Installation & Operation Manual Volume 2*, Canada.
- Harris,Ben (2007), *GPSTK user reference*, http://www.gpstk.org
- Stroustrup, Bjarne (1999), *Η Γλώσσα Προγραμματισμού C++*, AT&T,Murray Hill,New Jersey.
- Sedgewick,Robert (1998), *Algorithms in C++,*Addison Wesley, Princeton University.

ΠΑΡΑΡΤΗΜΑ Α

Ο κώδικας της εφαρμογής NORD σε C++.

Αρχεία ανάγνωσης/εγγραφής των παραμέτρων της εφαρμογής NORD

```
// ConfigFile.h


#ifndef CONFIGFILE_H
#define CONFIGFILE_H

#include <string>
#include <map>
#include <iostream>
#include <fstream>
#include <sstream>

using std::string;

class ConfigFile {
// Data
protected:
	string myDelimiter;  // separator between key and value
	string myComment;    // separator between value and comments
	string mySentry;     // optional string to signal end of file
	std::map<string,string> myContents;  // extracted keys and values

	typedef std::map<string,string>::iterator mapi;
	typedef std::map<string,string>::const_iterator mapci;

// Methods
public:
	ConfigFile( string filename,
		    string delimiter = "=",
		    string comment = "#",
			    string sentry = "EndConfigFile" );
	ConfigFile();

	// Search for key and read value or optional default value
	template<class T> T read( const string& key ) const;  // call as
read<T>
	template<class T> T read( const string& key, const T& value ) const;
	template<class T> bool readInto( T& var, const string& key ) const;
	template<class T>
	bool readInto( T& var, const string& key, const T& value ) const;

	// Modify keys and values
	template<class T> void add( string key, const T& value );
	void remove( const string& key );

	// Check whether key exists in configuration
	bool keyExists( const string& key ) const;

	// Check or change configuration syntax
	string getDelimiter() const { return myDelimiter; }
	string getComment() const { return myComment; }
	string getSentry() const { return mySentry; }
```

```
	string setDelimiter( const string& s )
		{ string old = myDelimiter;  myDelimiter = s;  return old; }
	string setComment( const string& s )
		{ string old = myComment;  myComment = s;  return old; }

	// Write or read configuration
	friend std::ostream& operator<<( std::ostream& os, const ConfigFile&
cf );
	friend std::istream& operator>>( std::istream& is, ConfigFile& cf );

protected:
	template<class T> static string T_as_string( const T& t );
	template<class T> static T string_as_T( const string& s );
	static void trim( string& s );


// Exception types
public:
	struct file_not_found {
		string filename;
		file_not_found( const string& filename_ = string() )
			: filename(filename_) {} };
	struct key_not_found {  // thrown only by T read(key) variant of
read()
		string key;
		key_not_found( const string& key_ = string() )
			: key(key_) {} };
};


/* static */
template<class T>
string ConfigFile::T_as_string( const T& t )
{
	// Convert from a T to a string
	// Type T must support << operator
	std::ostringstream ost;
	ost << t;
	return ost.str();
}


/* static */
template<class T>
T ConfigFile::string_as_T( const string& s )
{
	// Convert from a string to a T
	// Type T must support >> operator
	T t;
	std::istringstream ist(s);
	ist >> t;
	return t;
}


/* static */
template<>
inline string ConfigFile::string_as_T<string>( const string& s )
{
```

```
	// Convert from a string to a string
	// In other words, do nothing
	return s;
}


/* static */
template<>
inline bool ConfigFile::string_as_T<bool>( const string& s )
{
	// Convert from a string to a bool
	// Interpret "false", "F", "no", "n", "0" as false
	// Interpret "true", "T", "yes", "y", "1", "-1", or anything else as
true
	bool b = true;
	string sup = s;
	for( string::iterator p = sup.begin(); p != sup.end(); ++p )
		*p = toupper(*p);  // make string all caps
	if( sup==string("FALSE") || sup==string("F") ||
	    sup==string("NO") || sup==string("N") ||
	    sup==string("0") || sup==string("NONE") )
		b = false;
	return b;
}


template<class T>
T ConfigFile::read( const string& key ) const
{
	// Read the value corresponding to key
	mapci p = myContents.find(key);
	if( p == myContents.end() ) throw key_not_found(key);
	return string_as_T<T>( p->second );
}


template<class T>
T ConfigFile::read( const string& key, const T& value ) const
{
	// Return the value corresponding to key or given default value
	// if key is not found
	mapci p = myContents.find(key);
	if( p == myContents.end() ) return value;
	return string_as_T<T>( p->second );
}


template<class T>
bool ConfigFile::readInto( T& var, const string& key ) const
{
	// Get the value corresponding to key and store in var
	// Return true if key is found
	// Otherwise leave var untouched
	mapci p = myContents.find(key);
	bool found = ( p != myContents.end() );
	if( found ) var = string_as_T<T>( p->second );
	return found;
}
```

```
template<class T>
bool ConfigFile::readInto( T& var, const string& key, const T& value )
const
{
	// Get the value corresponding to key and store in var
	// Return true if key is found
	// Otherwise set var to given default
	mapci p = myContents.find(key);
	bool found = ( p != myContents.end() );
	if( found )
		var = string_as_T<T>( p->second );
	else
		var = value;
	return found;
}


template<class T>
void ConfigFile::add( string key, const T& value )
{
	// Add a key with given value
	string v = T_as_string( value );
	trim(key);
	trim(v);
	myContents[key] = v;
	return;
}

#endif  // CONFIGFILE_H

// Triplet.h


#include <iostream>

struct Triplet
{
	int a, b, c;

	Triplet() {}
	Triplet( int u, int v, int w ) : a(u), b(v), c(w) {}
	Triplet( const Triplet& orig ) : a(orig.a), b(orig.b), c(orig.c) {}

	Triplet& operator=( const Triplet& orig )
		{ a = orig.a;  b = orig.b;  c = orig.c;  return *this; }
};


std::ostream& operator<<( std::ostream& os, const Triplet& t )
{
	// Save a triplet to os
	os << t.a << " " << t.b << " " << t.c;
	return os;
}


std::istream& operator>>( std::istream& is, Triplet& t )
{
```

```
	// Load a triplet from is
	is >> t.a >> t.b >> t.c;
	return is;
}

// ConfigFile.cpp

#include "ConfigFile.h"

using std::string;

ConfigFile::ConfigFile( string filename, string delimiter,
                        string comment, string sentry )
	: myDelimiter(delimiter), myComment(comment), mySentry(sentry)
{
	// Construct a ConfigFile, getting keys and values from given file

	std::ifstream in( filename.c_str() );

	if( !in ) throw file_not_found( filename );

	in >> (*this);
}


ConfigFile::ConfigFile()
	: myDelimiter( string(1,'=') ), myComment( string(1,'#') )
{
	// Construct a ConfigFile without a file; empty
}


void ConfigFile::remove( const string& key )
{
	// Remove key and its value
	myContents.erase( myContents.find( key ) );
	return;
}


bool ConfigFile::keyExists( const string& key ) const
{
	// Indicate whether key is found
	mapci p = myContents.find( key );
	return ( p != myContents.end() );
}


/* static */
void ConfigFile::trim( string& s )
{
	// Remove leading and trailing whitespace
	static const char whitespace[] = " \n\t\v\r\f";
	s.erase( 0, s.find_first_not_of(whitespace) );
	s.erase( s.find_last_not_of(whitespace) + 1U );
}


std::ostream& operator<<( std::ostream& os, const ConfigFile& cf )
```

```
{
	// Save a ConfigFile to os
	for( ConfigFile::mapci p = cf.myContents.begin();
	     p != cf.myContents.end();
	        ++p )
	{
	      os << p->first << " " << cf.myDelimiter << " ";
	      os << p->second << std::endl;
	}
	return os;
}


std::istream& operator>>( std::istream& is, ConfigFile& cf )
{
	// Load a ConfigFile from is
	// Read in keys and values, keeping internal whitespace
	typedef string::size_type pos;
	const string& delim  = cf.myDelimiter;  // separator
	const string& comm   = cf.myComment;    // comment
	const string& sentry = cf.mySentry;     // end of file sentry
	const pos skip = delim.length();        // length of separator

	string nextline = "";  // might need to read ahead to see where
value ends

	while( is || nextline.length() > 0 )
	{
	      // Read an entire line at a time
	      string line;
	      if( nextline.length() > 0 )
	      {
	            line = nextline;  // we read ahead; use it now
	            nextline = "";
	      }
	      else
	      {
	            std::getline( is, line );
	      }

	      // Ignore comments
	      line = line.substr( 0, line.find(comm) );

	      // Check for end of file sentry
	      if( sentry != "" && line.find(sentry) != string::npos ) return
is;

	      // Parse the line if it contains a delimiter
	      pos delimPos = line.find( delim );
	      if( delimPos < string::npos )
	      {
	            // Extract the key
	            string key = line.substr( 0, delimPos );
	            line.replace( 0, delimPos+skip, "" );

	            // See if value continues on the next line
	            // Stop at blank line, next line with a key, end of
stream,
	            // or end of file sentry
```

```
                    bool terminate = false;
                    while( !terminate && is )
                    {
                           std::getline( is, nextline );
                           terminate = true;

                           string nlcopy = nextline;
                           ConfigFile::trim(nlcopy);
                           if( nlcopy == "" ) continue;

                           nextline = nextline.substr( 0,
nextline.find(comm) );
                           if( nextline.find(delim) != string::npos )
                                  continue;
                           if( sentry != "" && nextline.find(sentry) !=
string::npos )
                                  continue;

                           nlcopy = nextline;
                           ConfigFile::trim(nlcopy);
                           if( nlcopy != "" ) line += "\n";
                           line += nextline;
                           terminate = false;
                    }

                    // Store key and value
                    ConfigFile::trim(key);
                    ConfigFile::trim(line);
                    cf.myContents[key] = line;  // overwrites if key is
repeated
             }
       }

       return is;
}
```

Δομές (Structs) μυνημάτων (log) δέκτη

```
#ifndef __OEM4_BINARY_HEADER_H
#define __OEM4_BINARY_HEADER_H
struct OEM4_BINARY_HEADER         // Standard binary header
{
   unsigned char          sop1;          // start of packet first byte
   unsigned char          sop2;          // start of packet second byte
   unsigned char          sop3;          // start of packet third  byte
   unsigned char          header_length; // Length of the header
   unsigned short         number;        // Message number
   unsigned char          type;          // Message type
   unsigned char          port_address;  // Address of the data port the
log was received on
   unsigned short         length;        // Message length
   unsigned short         sequence;      // Sequence #
   unsigned char          idle;          // Idle time
   unsigned char          gps_stat;      // GPS Time Status
   unsigned short         gps_week;      // GPS Week number
   unsigned long          millisecs;     // Milliseconds into week
   unsigned long          status;        // Receiver status word
   unsigned short         crc;           // The 16bit CRC check sum
   unsigned short         version;       // Receiver software version
};
#endif

#include "header.h"

typedef struct
{
   OEM4_BINARY_HEADER     hdr;
   unsigned long          iObs;        // Number of observations
   unsigned char          data[24];    // packed binary range data
} RANGECMPB_LOG;

#include "header.h"

typedef struct
{
   unsigned long  ulPRN;               // satellite ID number
   unsigned long  ulTimeWeek;          // almanac reference time
weeks
   double         dTimeSec;            // almanac reference time
sec
   double         dEcc;                // eccentricity
   double         dRateRA;             // rate of right ascension
rad/sec
   double         dRA;                 // right ascension
rad
   double         dW;                  // argument of perigee
rad
   double         dM0;                 // mean anomaly
rad
   double         dCAf0;               // clock aging parameter
sec
   double         dCAf1;               // clock aging parameter
sec/sec
```

```
   double         dN;                 // corrected mean motion
rad/sec
   double         dA;                 // semi-major axis
m
   double         dInclA;             // angle of inclination
rad
   unsigned long  ulSVConfig;         // Satellite Configuration
   unsigned long  ulHealthPrn;        // SV health (subframe 4 or 5)
   unsigned long  ulHealthAlm;        // SV health (almanac)
   int            bAntiSpoof;         // Anti-Spoofing On
} ALMANACB_DATA;

typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   long                    lNumMessages;     // Number of satellite PRN
almanac messages to follow
   ALMANACB_DATA           data[32];
} ALMANACB_LOG;

#include "header.h"

typedef struct
{
   unsigned long  prn;              // Satellite PRN number
   unsigned long  ereferweek;       // Ephemeris Reference Week
   unsigned long  erefertime;       // Ephemeris Reference Time
 unsigned  char   subframe1[30];    // Ephemeris subframe 1
 unsigned  char   subframe2[30];    // Ephemeris subframe 2
 unsigned  char   subframe3[30];    // Ephemeris subframe 3
   unsigned short filler1;
   unsigned short filler2;
} RAWEPHEM_DATA;

typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   RAWEPHEM_DATA           data;
} RAWEPHEM_LOG;

#include "header.h"

#define MAXCHAN   24      // Maximum number of signal channels

typedef struct
{
   unsigned short svprn;        // PRN
   unsigned short frequency;    // Frequency number of GLONASS SV (0 for
GPS)
   double         psr;          // pseudo range
   float          psrstd;       // pseudorange standard deviation
   double         adr;          // accumulated doppler
   float          adrstd;       // accumulated doppler standard deviation
   float          dop;          // Doppler
   float          sno;          // Signal/Noise
   float          locktime;     // time locked
   unsigned long  ch_status;    // channel status
} RANGEB_DATA;
```

```
typedef struct
{
   OEM4_BINARY_HEADER  hdr;
   unsigned long  iObs;           // Number of observations
   RANGEB_DATA         data[MAXCHAN];
} RANGEB_LOG;

#include "header.h"

typedef struct
{
   unsigned long  sol_stat;           // Solution status
   unsigned long  pos_type;           // Position type
   double         lat;                // Latitude
   double         lon;                // Longitude
   double         hgt;                // Height above mean sea level
   float          und;                // Undulation
   long           datum_id;           // Datum ID
   float          latdev;             // Latitude standard deviation
   float          londev;             // Longitude standard deviation
   float          hgtdev;             // Height standard deviation
   char           ref_station_id[4];  // Reference station ID
   float          lag;                // Differential correction lag
   float          age;                // Solution age
   unsigned char  sats;               // Number of matched satellites
   unsigned char  gps_l1_ranges;      // Number of GPS L1 ranges used
   unsigned char  gps_l1_mask_ranges; // Number of GPS L1 ranges used
   unsigned char  gps_l2_mask_ranges; // Number of GPS L2 ranges used
   char           reserved[4];
} OEM4POSB_LOG;

// BESTPOS log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4POSB_LOG            data;
} BESTPOSB_LOG;

// MARKPOS log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4POSB_LOG            data;
} MARKPOSB_LOG;

// MATCHEDPOS log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4POSB_LOG            data;
} MATCHEDPOSB_LOG;

// PSRPOS log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4POSB_LOG            data;
} PSRPOSB_LOG;
```

```
// RTKPOS log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4POSB_LOG            data;
} RTKPOSB_LOG;

#include "header.h"

#define  COM_PASSTHROUGH_MAX  (80)        // Maximum size of the pass
through log

typedef struct
{
   OEM4_BINARY_HEADER   hdr;
   unsigned long        number_bytes;                  // Number of bytes
to follow
   char                 my_buff[COM_PASSTHROUGH_MAX];   // Message data
} PASSCOMB_LOG;

#include "header.h"

typedef struct
{
   OEM4_BINARY_HEADER   hdr;
   unsigned long        ulSolStatus;     // Solution status
   unsigned long        ulPosType;       // Position type
   unsigned long        ulVelType;       // Velocity type
   unsigned long        ulNavType;       // Navigation data type
   double               dDistance;       // See the OEM4 manual for a
detailed description of the following fields
   double               dBearing;
   double               dAlongTrack;
   double               dXTrack;
   int                  iEtaWeek;
   double               dEtaSeconds;
} NAVIGATEB_LOG;

typedef struct
{
   unsigned long  prn;              // Satellite PRN number
   double tow;
   unsigned long health;
   unsigned long iode1;
   unsigned long iode2;
   unsigned long week;
   unsigned long zweek;
   double toe;
   double a;
   double dn;
   double m0;
   double ecc;
   double w;
   double cuc;
   double cus;
   double crc;
   double crs;
   double cic;
   double cis;
```

```
   double i0;
   double io;
   double w0;
   double wo;
   unsigned long iodc;
   double toc;
   double tgd;
   double af0;
   double af1;
   double af2;
   unsigned long as;
   double n;
   double ura;
   } RAWEPHEM_DATA;

typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   RAWEPHEM_DATA           data;
} RAWEP

#include "header.h"

typedef struct
{
   unsigned long  ulClockModelStatus;        // Clock model status
   unsigned long  ulRejectCount;             // Number of rejected range
bias measurements
   unsigned long  ulNoiseMillisecs;          // GPS time of last noise
addition since Jan 3, 1980
   unsigned long  ulUpdateMillisecs;         // GPS time of last update
since Jan 3,
   double         dRangeBias;                // Range bias (clock
correction parameters)
   double         dRangeBiasRate;            // Range bias rate (clock
correction parameters)
   double         dSAB;                      // Range bias error due to SA
(clock correction parameters)
   double         dCovXX;                    // ---------------------------
--------
   double         dCovXY;                    //
   double         dCovXZ;                    //
   double         dCovYX;                    //
   double         dCovYY;                    //     The covariance matrix
   double         dCovYZ;                    //
   double         dCovZX;                    //
   double         dCovZY;                    //
   double         dCovZZ;                    // ---------------------------
---------
   double         dInstRangeBias;            // Instantaneous range bias
(m)
   double         dInstDrift;                // Instantaneous drift
measurement (m/s)
   unsigned long  ulConstellationChange;     // Constellation change
(1=yes, 0=no)
} CLOCKMODELB_BODY;

typedef struct
{
```

```
   OEM4_BINARY_HEADER     hdr;
   CLOCKMODELB_BODY       body;
} CLOCKMODELB_LOG;

#include "header.h"

enum POS_AVE_STAT
{
   OFF_AVE = 0,    // Card is not averaging
   INPROGRESS,     // Averaging is currently in progress
   COMPLETE        // Averaging is complete
};

typedef struct
{
   OEM4_BINARY_HEADER  hdr;
   double              dLat;         // Average WGS84 latitude (degrees)
   double              dLon;         // Average WGS84 longitude (degrees)
   double              dHgt;         // Average height above sea level (m)
   float               fLatDev;      // Estimated average standard
deviation of WGS84 latitude (degrees)
   float               fLonDev;      // Estimated average standard
deviation of WGS84 longitude (degrees)
   float               fHgtDev;      // Estimated average standard
deviation of height (m)
   POS_AVE_STAT        eSolnstat;    // Solution status corresponding to
the POSITION AVERAGING TABLE in the manual
   unsigned long       ulAvetime;    // Elapsed time of everaging (s)
   unsigned long       ulSamples;    // Number of samples in the average
} AVEPOSB_LOG;

#include "header.h"


#define MAX_COMPONENTS       10     // This is the current limit in the
OEM4 card

#define BIN_MODEL_LEN        16
#define BIN_PSN_LEN          16
#define BIN_HWVERSION_LEN    16
#define BIN_SWVERSION_LEN    16
#define BIN_BOOTVERSION_LEN  16
#define BIN_COMPDATE_LEN     12
#define BIN_COMPTIME_LEN     12
#define BIN_MAX_LEN          16     // The maximum of the above lengths

typedef struct
{
   unsigned long  comptype;                          // Component type
   char           model[BIN_MODEL_LEN];              // Model
   char           psn[BIN_PSN_LEN];                  // Serial number
   char           hwversion[BIN_HWVERSION_LEN];      // Hardware version
   char           swversion[BIN_SWVERSION_LEN];      // Software version
   char           bootversion[BIN_BOOTVERSION_LEN];  // Boot software
version
   char           compdate[BIN_COMPDATE_LEN];        // Compile date
   char           comptime[BIN_COMPTIME_LEN];        // Compile time
} VERSIONB_DATA;
```

```
typedef struct
{
   OEM4_BINARY_HEADER     hdr;
   unsigned long                  components;            // Number of
components in the log
   VERSIONB_DATA          data[MAX_COMPONENTS];
} VERSIONB_LOG;

#include "header.h"

typedef struct
{
   unsigned long  vel_status;       // velocity status (solution status)
   unsigned long  vel_type;         // velocity type (Solution type)
   float          latency;          // velocity log latency
   float          lag;              // differential correction age
   double         hor_speed;       // horizontal velocity
   double         trk_gnd;         // direction of travel over ground
   double         vert_speed;      // vertical velocity
   float          std;              // 3d standard deviation
} OEM4VELB_LOG;

// BESTVEL log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4VELB_LOG            data;
} BESTVELB_LOG;

// PSRVEL log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4VELB_LOG            data;
} PSRVELB_LOG;

// RTKVEL log structure
typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   OEM4VELB_LOG            data;
} RTKVELB_LOG;

#include "header.h"

#define MAXCHAN   24               // Maximum number of signal channels

typedef struct
{
   unsigned long  ulsol_status;            // Solution status
   unsigned long  ulpos_type;              // Position type
   float          ftrack_elev_cutoff;      // Tracking elevation cutoff
angle
   unsigned long  ulnumber_of_chans;       // Number of hardware channels
with information to follow
} TRACKSTATB_HEADER;

typedef struct
{
```

```
   unsigned short ulPRN;                   // Satellite PRN number
   unsigned short ulFrequency;             // Glonass Frequency index
   unsigned long  ultracking_status;       // Channel tracking status
   double         dpseudorange;            // Pseudorange (m)
   float          fdoppler_freq;           // Doppler frequency (Hz)
   float          fCNo;                    // Carrier to noise density
ratio (dB-Hz)
   float          flocktime;               // Number of seconds of
continuous tracking
   float          fPSR_residual;           // Pseudorange residual from
pseudorange filter (m)
   unsigned long  ulPSR_range_reject_code; // Range reject code from
pseudorange filter (m)
   float          fPSR_Filter_weight;      // Pseudorange filter weighting
} TRACKSTATB_DATA;


typedef struct
{
   OEM4_BINARY_HEADER   hdr;
   TRACKSTATB_HEADER    header;
   TRACKSTATB_DATA      data[MAXCHAN];
}TRACKSTATB_LOG;

#include "header.h"

typedef struct
{
   unsigned long  ulClockModel;   // ClockModelStatus
   double         dGPSOffset;     // Receiver Offset in seconds from GPS
time
   double         dOffsetStd;     // Instantaneous Standard Deviation of
Receiver Clock Offset
   double         dUtcOffset;     // Offset in seconds of GPS time from UTC
time
   long           lUtcYear;       // UTC Year
   unsigned char  ucUtcMonth;     // UTC Month
   unsigned char  ucUtcDay;       // UTC Day
   unsigned char  ucUtcHour;      // UTC Hour
   unsigned char  ucUtcMin;       // UTC Minutes
   long           lUtcMillisec;   // UTC Milliseconds
   int            bUtcStatus;     // UTC Status
} TIME_DATA;

typedef struct
{
   OEM4_BINARY_HEADER   hdr;
   TIME_DATA            body;
} TIMEB_LOG;

#include "header.h"

#define MAXSVCHAN 24               // Maximum number of SVs that we track
at one time

typedef struct
{
   double         dReserved1;
   unsigned long  ulNumSats;              // number of satellites
```

```
} SATXYZB_HEADER;

typedef struct
{
   unsigned long  ulPRN;                  // SV PRN
   double         dX;                     // SV X coordinate (metres)
   double         dY;                     // SV Y coordinate (metres)
   double         dZ;                     // SV Z coordinate (metres)
   double         dClkCorr;               // SV clock correction
   double         dIonCorr;               // ionospheric correction
   double         dTropCorr;              // tropospheric correction
   double         dReserved2;             // reserved
   double         dReserved3;             // reserved
} SATXYZB_DATA;

typedef struct
{
   OEM4_BINARY_HEADER   oem4hdr;
   SATXYZB_HEADER       header;
   SATXYZB_DATA         data[MAXSVCHAN];
} SATXYZB_LOG;

#include "header.h"
#define MAX_NUM_SAT 50

typedef struct
{
   int            bSatVis;             // Is satellite visibility valid?
   int            bCompAlm;            // Was complete almanac used?
   unsigned long  ulNumSat;            // Number of satellites with
information in log
} SATVISB_HEADER;

typedef struct
{
   short          sPrn;                // Satellite PRN number
   short          sReserved;
   unsigned long  ulHealth;            // Satellite health
   double         dElev;               // Satellite elevation (degrees)
   double         dAz;                 // Satellite azimuth (degrees)
   double         dTrueDop;            // Theoretical doppler of satellite
   double         dAppDop;             // Apparent doppler of satellite
} SATVISB_DATA;

typedef struct
{
   OEM4_BINARY_HEADER      hdr;
   SATVISB_HEADER          header;
   SATVISB_DATA            data[MAX_NUM_SAT];
} SATVISB_LOG;

#include "header.h"

#define MAXSTATS 4       // Maximum number of status codes contained in the
log

typedef struct
{
   unsigned long  ulError;                      // Reciever Error
```

```
   unsigned long  ulNumStats;                   // Number of Status Codes
} RXSTATUSB_HEADER;

typedef struct
{
   unsigned long  ulRxstat;                  // Reciever Status Word
   unsigned long  ulRxstatPri;               // Reciever Status Priority
Mask
   unsigned long  ulRxstatSet;               // Reciever Status Event Set
Mask
   unsigned long  ulRxstatClear;             // Reciever Status Event Clear
Mask
} RXSTATUSB_DATA;

typedef struct
{
   OEM4_BINARY_HEADER     hdr;
   RXSTATUSB_HEADER       header;
   RXSTATUSB_DATA         data[MAXSTATS];
} RXSTATUSB_LOG;
```

Αρχεία υλοποίησης Socket

```
// Definition of the ServerSocket class

#ifndef ServerSocket_class
#define ServerSocket_class

#include "Socket.h"


class ServerSocket : private Socket
{
 public:

  ServerSocket ( int port );
  ServerSocket (){};
  virtual ~ServerSocket();

  const ServerSocket& operator << ( const std::string& ) const;
  const ServerSocket& operator >> ( std::string& ) const;

  void accept ( ServerSocket& );

};


#endif

// Implementation of the ServerSocket class

#include "ServerSocket.h"
#include "SocketException.h"


ServerSocket::ServerSocket ( int port )
{
  if ( ! Socket::create() )
    {
      throw SocketException ( "Could not create server socket." );
    }

  if ( ! Socket::bind ( port ) )
    {
      throw SocketException ( "Could not bind to port." );
    }

  if ( ! Socket::listen() )
    {
      throw SocketException ( "Could not listen to socket." );
    }

}

ServerSocket::~ServerSocket()
{
}
```

```
const ServerSocket& ServerSocket::operator << ( const std::string& s )
const
{
  if ( ! Socket::send ( s ) )
    {
      throw SocketException ( "Could not write to socket." );
    }

  return *this;

}


const ServerSocket& ServerSocket::operator >> ( std::string& s ) const
{
  if ( ! Socket::recv ( s ) )
    {
      throw SocketException ( "Could not read from socket." );
    }

  return *this;
}

void ServerSocket::accept ( ServerSocket& sock )
{
  if ( ! Socket::accept ( sock ) )
    {
      throw SocketException ( "Could not accept socket." );
    }
}

// Definition of the ClientSocket class

#ifndef ClientSocket_class
#define ClientSocket_class

#include "Socket.h"


class ClientSocket : private Socket
{
 public:

  ClientSocket ( std::string host, int port );
  virtual ~ClientSocket(){};

  const ClientSocket& operator << ( const std::string& ) const;
  const ClientSocket& operator >> ( std::string& ) const;

};


#endif

// Implementation of the ClientSocket class

#include "ClientSocket.h"
#include "SocketException.h"
```

```
ClientSocket::ClientSocket ( std::string host, int port )
{
  if ( ! Socket::create() )
    {
      throw SocketException ( "Could not create client socket." );
    }

  if ( ! Socket::connect ( host, port ) )
    {
      throw SocketException ( "Could not bind to port." );
    }

}


const ClientSocket& ClientSocket::operator << ( const std::string& s )
const
{
  if ( ! Socket::send ( s ) )
    {
      throw SocketException ( "Could not write to socket." );
    }

  return *this;

}


const ClientSocket& ClientSocket::operator >> ( std::string& s ) const
{
  if ( ! Socket::recv ( s ) )
    {
      throw SocketException ( "Could not read from socket." );
    }

  return *this;
}

// Definition of the Socket class

#ifndef Socket_class
#define Socket_class


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string>
#include <arpa/inet.h>


const int MAXHOSTNAME = 200;
const int MAXCONNECTIONS = 5;
const int MAXRECV = 500;

class Socket
```

```
{
 public:
  Socket();
  virtual ~Socket();

  // Server initialization
  bool create();
  bool bind ( const int port );
  bool listen() const;
  bool accept ( Socket& ) const;

  // Client initialization
  bool connect ( const std::string host, const int port );

  // Data Transimission
  bool send ( const std::string ) const;
  int recv ( std::string& ) const;


  void set_non_blocking ( const bool );

  bool is_valid() const { return m_sock != -1; }

 private:

  int m_sock;
  sockaddr_in m_addr;


};


#endif

// SocketException class


#ifndef SocketException_class
#define SocketException_class

#include <string>

class SocketException
{
 public:
  SocketException ( std::string s ) : m_s ( s ) {};
  ~SocketException (){};

  std::string description() { return m_s; }

 private:

  std::string m_s;

};

#endif
```

Αρχεία κλάσεων NORD

```
#ifndef __BUFFER_HPP
#define __BUFFER_HPP
#include <string.h>
#include <limits.h>
#include "Defines.h"
#include "ncexcept.hpp"
#define  BUFFER_MAX_SIZE   (8192)      // The maximum size of buffer that
can be given

enum BUFFER_SET_RETURN
{
  BUFFER_SET_GOOD,         // All the data has been taken
  BUFFER_SET_NOT_ENOUGH_ROOM,    // Not enough room in the serial input.
Call Set again before destructing the ucBuf_ data.
  BUFFER_SET_REMAINS
};

enum BUFFER_GET_RETURN
{
  BUFFER_GET_GOOD,         // All the data has been given
  BUFFER_GET_EMPTY         // There is not enough data to be taken
};

class nBuffer
{
protected:
  unsigned char ucBuff[BUFFER_MAX_SIZE * 2];  // Double the size character
buffer

  int iOffset;                   // The offset marked by the user
  int iBuffStart;          // Where the data begins
  int iBuffEnd;                  // Where the data ends

  long lTotalSize;         // The sum of all the reads
  bool bMyWrapped;         // Indicates lTotalSize has wrapped

public:
    nBuffer ();

  // Fill the buffer
  BUFFER_SET_RETURN Set (unsigned char *ucBuf_, int iSize_);

  // This method makes the requested number of bytes available to be
  //                          // written to by the caller.  This method does
NOT set the data,
  //                               // it simply sets aside the requested
space in nBuffer's internal buffer,
  //                                     // and returns a pointer to the
start of the buffer area set aside.
  //                                           // Returns NULL if there is
not enough space to be set aside.
  unsigned char *SetRequestBufSpace (int iSizeRequested_);
  // Fill the buffer with a string (also copy the nul)
  BUFFER_SET_RETURN Puts (char *szStr_);
  // Read from the buffer, if ucBuf_ is NULL no data is transfered
  BUFFER_GET_RETURN Get (unsigned char *ucBuf_, int iSize_);
```

```
  BUFFER_GET_RETURN Get (unsigned char *ucBuf_, int iSize_, unsigned char
ch, short *iNumBytes);    // Read up to ch
  // What is the current offset?
  long Tell ();
  // Record the offset into ucBuff
  void SetSync ();
  // Go back to the offset into ucBuff
  void Sync ();
  // Go to the next character to find data to decode
  void ReSync (int iSize = 1);
  // Clear the buffer and stats
  void Clear ();
  // Get the amount in the ucBuff
  int iAmount ()
  {
    return (iBuffEnd - iBuffStart);
  }
};
#endif

// Filename:   NDefines.h

#ifndef __NDEFINES_H
#define __NDEFINES_H

////////////////////////////////////////////////////////////////////
// Standard Types
////////////////////////////////////////////////////////////////////
typedef  signed int     INT;     // Signed Machine word size   (Assume to
be no larger than 16 bit)
typedef  unsigned int   UINT;    // Unsigned Machine word size (Assume to
be no larger than 16 bit)

// Defined by microsoft
typedef  char           CHAR;    // Signed 1-byte
typedef  unsigned char  UCHAR;   // Unsigned 1-byte
typedef  signed short   SHORT;   // Signed 2-byte (Unsuported for the T-
805)
typedef  unsigned short USHORT;  // Unsigned 2-byte (Unsuported for the T-
805)
typedef  signed long    LONG;    // Signed 4-byte
typedef  unsigned long  ULONG;   // Unsigned 4-byte

typedef  float          FLOAT;   // Single precision - IEEE floating point
32 bits
typedef  double         DOUBLE;  // Double precision - IEEE floating point
64 bits
typedef  long double    LDOUBLE; // Extended double precision - 80 bits

typedef  int            BOOL;    // BOOL (TRUE or FALSE)
typedef  unsigned char  BOOLCHAR;// BOOL (TRUE or FALSE) 1-byte


////////////////////////////////////////////////////////////////////
// Other Global Types
////////////////////////////////////////////////////////////////////
typedef  DOUBLE         GPS_TIME;   // Time in seconds
typedef  LDOUBLE        BTIME;      // Big Time in bigger seconds
```

```
//////////////////////////
// BOOLEAN Values
//////////////////////////
#define TRUE        (1)
#define FALSE       (0)
#define YES         (1)
#define NO          (0)
#define ON          (1)
#define OFF         (0)
#define GOOD        (1)
#define BAD         (0)

//////////////////////////
// NULLs
//////////////////////////
#ifndef  NULL
#define  NULL  (0)    // Pointer
#endif
#ifndef  NUL
#define  NUL   (0)    // End of String
#endif

#define DBL_EQUAL(x,v) ((fabs((DOUBLE) (x) - (DOUBLE) (v) )) < DBL_EPSILON)

// Make character upper case
#define  TOUPPER(ch)    ((ch) >= 'a' && (ch) <= 'z' ? (ch)-'a'+'A' : (ch))

// Convert an ASCII hex character to a nibble
#define  CHARTOHEX(ch)  ((ch) >= 'A' ? ((ch) - 'A')+10 : (ch) - '0')

// 08/12/1999 [CEL]  I added a char in front of the definition for
HEXTOCHAR as
//                   it was generating compiler errors in the form of
//                   significant figures may be lost due to conversion.
// Convert a nibble to a ASCII hex character
#define  HEXTOCHAR(hx)  char((hx) >= 0x0A ? (hx) + 'A' - 0x0A : (hx) + '0')

// Convert a nibble to a ASCII hex character
#define  ISHEX(hx)  (((hx) >= '0' && (hx) <= '9') || (TOUPPER(hx) >= 'A' &&
TOUPPER(hx) <= 'F'))

// Return the smallest of the two values
#define MIN(x,y)  ((x)<(y) ? (x) : (y))

// Return the largest of the two values
#define MAX(x,y)  ((x)<(y) ? (y) : (x))

// Return the square value
#define SQR(x) ((x)*(x))

// Compare DOUBLES
#define COMPARE_REAL(X, Y, D) ((fabs((X)-(Y)) < (D)) ? 1:0)

// Conversion factors
#define M2INM    ((double) (0.00053995680))  /* INM - International
nautical mile */
#define M2FT     ((double) (1.0/0.3048))
#define M2MILE   ((double) (0.00062137119))
#define MBAR2KPA ((double) (1.0/10.0))
```

```
#define MBAR2INI ((double) (1.0/33.8639))
#define MBAR2MM  ((double) (1.0/1.33322))
#define MBAR2PSI ((double) (1.0/68.94757))

// Hex conversion routines; definition ONLY.  Must include "hexconv.cpp" to
use.
unsigned long htol( const char *pszStringIn );
void htoArray( const char *pszStringIn , char* array, int iMaxNum);
void Arraytoh( const char *ArrayIn , char* pszStringOut, int iNumBytes);

#endif

//filename GpsOut.hpp
#ifndef __GpsOut_HPP
#define __GpsOut_HPP

#include "Defines.h"
#include "Buffer.hpp"
//#include "DBgps.hpp"
enum LOG_TYPE              // Used by iType
{
  UNKNOWN_LOG_TYPE,        // Unknown logs type
  NOVATEL_ASCII,           // Novatel ASCII logs
  NOVATEL_BINARY,          // Novatel Binary logs
  NOVATEL_OEM4_ASCII,            // Novatel ASCII logs
  NOVATEL_OEM4_BINARY,           // Novatel Binary logs
  NOVATEL_OEM4_ABBREVIATED_ASCII,      // Novatel Abbreviated ASCII logs
  NMEA,                          // National Marine Electronics Association
logs
  RINEX,                   // Rinex ASCII log
  CARD_RESPONSE,           // GPSCard response statement
  ERROR_PROMPT,                  // Prompt for an error message
  ACK_PROMPT,              // Prompt for acknowledge messages (eg. <OK)
  STANDARD_PROMPT
};
#define MAX_NOUT_SIZE      (8192)      // Maximum size of a NovAtel log
buffer (ALMANACA logs are big!)
#define NOUT_LARGE_UNKNOWN (20)  // Consider "large" unknown if bigger than
this
// !!!!!  Make sure that when you add a new log, it is between the
LOGS_START and LOGS_END entries of the appropriate derived parsing class
// // !!!!!  When changing this table make sure changes are also made to
g_astLogMap in NoutUnknown.CPP
enum NOUT_ID
{
  UNKNOWN,


  LOGS_START,
  ACPA,
  AGCA,
  ALMA,
  ATTA,
  BATA,
  BSLA,
  CALA,
  CDSA,
  CLKA,
  CLMA,
```

```
COM1A,
COM2A,
CONSOLEA,
CORA,
CTSA,
DCSA,
DIRA,
DOPA,
ETSA,
FRMA,
FRWA,
GALA,
GCLA,
GEPA,
GROUPA,
GRPA,
HDRA,
IONA,
ISMRA,
KPHA,
LPSTATUSA,
META,
MKPA,
MKTA,
MPMA,
MSGA,
NAVA,
OPTA,
P20A,
PAVA,
PDCA,
PDCDBG1A,
PDCVERA,
PNTA,
POSA,
PROJECTA,
PRTKA,
PSNA,
PXYA,
PVAA,
RALA,
RASA,
RBTA,
RCCA,
RCSA,
REPA,
RGEA,
RNGA,
RPSA,
RT20A,
RTCA,
RTCMA,
RTCM16T,
RTKA,
RTKOA,
RVSA,
SATA,
SBLA,
SBTA,
```

```
SCHA,
SFDA,
SITELOGA,
SNOA,
SPHA,
STATUSA,
SVDA,
TM1A,
UTCA,
VERA,
VLHA,
WALA,
WUTCA,
WBRA,
WRCA,

ACPB,
AGCB,
ALMB,
ATTB,
BATB,
BSLB,
CALB,
CDSB,
CLKB,
CLMB,
COM1B,
COM2B,
CONSOLEB,
CORB,
CTSB,
DCSB,
DIRB,
DLLB,
DOPB,
ETSB,
FRMB,
FRWB,
GALB,
GCLB,
GEPB,
GROUPB,
GRPB,
HDRB,
IONB,
ISMRB,
KPHB,
LPSTATUSB,
METB,
MKPB,
MKTB,
MPMB,
MSGB,
NAVB,
OPTB,
P20B,
PAVB,
PDCB,
PDCDBG1B,
```

```
PDCVERB,
POSB,
PROJECTB,
PRTKB,
PSNB,
PVAB,
PXYB,
RALB,
RASB,
RBTB,
RCSB,
REPB,
RGEB,
RGEC,
RGED,
RPSB,
RT20B,
RTCAB,
RTCMB,
RTKB,
RTKOB,
RVSB,
SATB,
SBLB,
SBTB,
SCHB,
SFDB,
SITELOGB,
SNOB,
SPHB,
STATUSB,
SVDB,
TM1B,
UTCB,
VERB,
VLHB,
WALB,
WUTCB,
WBRB,
WRCB,
SSOBSL1L2,
SSOBSL1,
SSOBSGISMO,
TAGB,
DICB,

GPALM,
GPGGA,
GPGLL,
GPGNS,
GPGRS,
GPGSA,
GPGST,
GPGSV,
GPRMB,
GPRMC,
GPVTG,
GPZDA,
GPZTG,
```

```
  GLALM,
  GLGGA,
  GLGLL,
  GLGNS,
  GLGRS,
  GLGSA,
  GLGST,
  GLGSV,
  GLRMB,
  GLRMC,
  GLVTG,
  GLZDA,
  GLZTG,
  GNALM,
  GNGGA,
  GNGLL,
  GNGNS,
  GNGRS,
  GNGSA,
  GNGST,
  GNGSV,
  GNRMB,
  GNRMC,
  GNVTG,
  GNZDA,
  GNZTG,

  PTNL_GGK,

  XOBS,
  XOHD,
  XNHD,
  XNAV,

  ZMESB,
  ZPOSB,
  ZEPHB,
  ZSTNB,
  ZCFGB,
  ZTAGB,

  TESTBED_TIMEA,
  TESTBED_TESTIDA,
  TESTBED_REPORTA,
  TESTBED_MIRRORA,
  TESTBED_TIMEB,
  TESTBED_TESTIDB,
  TESTBED_REPORTB,
  TESTBED_MIRRORB,

//  OEM4
  CLOCKMODELA,
  CLOCKMODELB,
  VERSIONA,
  VERSIONB,
  AVEPOSA,
  AVEPOSB,
  BESTPOSA,
  BESTPOSB,
```

```
BESTVELA,
BESTVELB,
MARKPOSA,
MARKPOSB,
MATCHEDPOSA,
MATCHEDPOSB,
NAVIGATEA,
NAVIGATEB,
PASSCOM1A,
PASSCOM1B,
PASSCOM2A,
PASSCOM2B,
PASSCOM3A,
PASSCOM3B,
PSRPOSA,
PSRPOSB,
PSRVELA,
PSRVELB,
PROPAGATEDCLOCKMODELA,
PROPAGATEDCLOCKMODELB,
RTKPOSA,
RTKPOSB,
RANGEA,
RANGEB,
RANGECMPA,
RANGECMPB,
RAWEPHEMA,
RAWEPHEMB,
RAWGPSSUBFRAMEA,
RAWGPSSUBFRAMEB,
REFSTATIONA,
REFSTATIONB,
RXCONFIGA,
RXCONFIGB,
RXSTATUSA,
RXSTATUSB,
SATSTATA,
SATSTATB,
TIMEA,
TIMEB,
TRACKSTATA,
TRACKSTATB,
ALMANACA,
ALMANACB,
IONUTCA,
IONUTCB,
CHANDEBUGA,
CHANDEBUGB,
UNKNOWNOEM4A,                 // Unknown but valid OEM4 ASCII log
UNKNOWNOEM4B,                 // Unknown but valid OEM4 binary log

POINTOBS,               // Point format NCObservation
POINTEPH,               // Point format NCOrbit

PASHR_POS,              // Ashtech
LOGS_END,
// End of log types
```

```
  // Start of Card responses
  FSU_TUNED,
  FSU_DETUNED,
  TRANSPARENTMODE_ON,
  TRANSPARENTMODE_OFF,

  COM1_PROMPT,
  COM2_PROMPT,
  COM3_PROMPT,
  COM1_PROMPT_LF,
  COM2_PROMPT_LF,
  COM3_PROMPT_LF,
  CONSOLE_PROMPT,
  CRLF_PROMPT,

  INVALID_COMMAND_NAME,
  INVALID_NUMBER_OF_ARGUMENTS,
  INVALID_DATA_LOGGER_TYPE,
  INVALID_COMMAND_OPTION,
  INVALID_IN_DATA,
  INVALID_CHANNEL_NUMBER,
  INVALID_SATELLITE_NUMBER,
  INVALID_DOPPLER,
  INVALID_DOPPLER_WINDOW,
  NVM_ERROR,
  OK_ACK,
  // End of Card responses

  // All identities flag  (IMPORTANT:  These should always be last in the
enum)
  MAX_NOUT_ID,
  ALL_NOUT_ID
};

class NOutUnknown
{
public:
  // Returned by NOutUnknown::Read
  enum RETURN
  {
    START,                // Start of Nout
    OKAY,                 // Everything is peachy keen
    EOFILE,               // End of file encountered
    EMPTY,                // Needs data
    RESYNC                // Moving to the next byte
  };

    NOutUnknown ();       // Constructor
    virtual ~ NOutUnknown ();    // Destructor

  // Give new data to serial buffer
  BUFFER_SET_RETURN Set (unsigned char *ucbuf, INT iSize);
  // Get the log buffer.  The Read is usually in the derived class
  UCHAR *Get ();
  // Defaults to returning the last log read identity as a string, log id
may be specified.
  const char *szIdStr (UINT iLogId = ALL_NOUT_ID)
  {
    return GetIDStr (iLogId);
```

```
  }

  // Return a printable synopsis of the logs read.
  // void PrintSum (NCString * ncStr);
  void Clear ();
  // Return the number of unidentified bytes.
  LONG Unknown ();
  // Return the number of blocks of unidentified bytes.
  ULONG ulNumLargeUnknownBytes ()
  {
    return ulMyNumLargeUnknownBytes;
  }
  // Return the number of logs of id read.
  INT iNumber (INT iId = ALL_NOUT_ID);

  // Information functions
  LOG_TYPE Type (void);          // Type of log
  NOUT_ID Id (void);             // Return the identity of the last read
log
  INT Size (void);         // Return size of last read buffer
  LONG TotalSize ();             // Return total size.
  LONG Tell ()                   // Return the local buffer offset
  {
    return SerDat.Tell ();
  }
  INT iBytesInRxBuf ()           // Number of unread bytes in serial buffer
are in the internal buffer
  {
    return SerDat.iAmount ();
  }
  ULONG ulGetiNumNOut (ULONG nId)
  {
    return iNumNOut[nId];
  }

  // Return the identity of the (usually) user entered log
  static NOUT_ID eIdStr (CHAR * szId);

  // Return the identity of the last read log
//  static NCString eIdStr (NOUT_ID eId);

  // Resync to the next byte
  void ReSync (int iSize = 1);

protected:
  // DATA
  nBuffer SerDat;          // Serializes blocks of data/memory
  RETURN OldNoutReturn;          // If the last read was good
  NOUT_ID iId;                   // Identity of the last log
  UCHAR ucBuf[MAX_NOUT_SIZE];    // Where the log is kept
  INT m_iSize;                   // The size of the last log
  LOG_TYPE iType;          // The type of the last log
  // DATA (statistical), not publicly accessible from this base class.
  ULONG ulUnknownBytes;          // Number of unknown bytes encountered
  ULONG ulLastUnknownBytes;      // Number of unknown bytes encountered
before the last log
  ULONG ulMyNumLargeUnknownBytes;      // Number of large unknown bytes
encountered
  LONG m_lTotalSize;             // The total accumulated size
```

```
  INT iNumNOut[MAX_NOUT_ID];     // Number of times each LogID has been
read.

  // FUNCTIONALITY
  INT CopyStr (nBuffer & dat);   // Copies Ascii characters from the file
  void DecodeAscii ();           // Identify Ascii log type, NovAtel or
NMEA
  void DecodeOEM4Ascii ();// Identify OEM4 Ascii log type
  void DecodeBinary ();          // Identify Binary log type
  void DecodeOEM4Binary ();      // Identify Binary log type

  const char *GetIDStr (UINT uiLogID = ALL_NOUT_ID);

private:
  // Helper function for PrintSum
//  void PrintSum_AddLog (NOUT_ID eId_, NCString * nsDestination_);

  // Array of string representations of nOutId (from NOutUnknown.hpp).
  char **m_ppszId;

};

class NXOut:public NCException
{
public:
 enum XCode
  {
    FunctionNotSupported = NXOUT_INDEX,// Function is not supported by the
parser
    InvalidID,                   // This is the original invalid id used by
Point
    InvalidLogId,          // This is what used to be the NExcept of Test
    EntryNotFound,         // Not in the table
    ReadForwardTwice             // I guess this isn't allowed
  };

  // constructors
    NXOut (XCode ExCode, char *szFile, int nLine, char *szDate,
          char *szTime):NCException (ExCode, szFile, nLine, szDate, szTime)
  {
  }

  NXOut (XCode ExCode, char *szFile, int nLine, char *szDate, char *szTime,
        char *szMsg):NCException (ExCode, szFile, nLine, szDate, szTime,
                                  szMsg)
  {
  }
};
#endif // GpsOut

//GpsOutC.hpp
#ifndef __GpsOutC_HPP
#define __GpsOutC_HPP
#include "GpsOut.hpp"
#include "logtypes.h"
#define OEM4_CRC_SIZE   (4)      // Number of bytes that makes up the CRC
#include "logheaders/range.h"
#include "logheaders/almanac.h"
#include "logheaders/satvis.h"
```

```
#include "logheaders/rawephem.h"
#include "logheaders/time.h"
#include "ncrc32.hpp"
#include "Stack.hpp"

#include <boost/any.hpp>
// GPSTk
#include <iostream>
#include <fstream>
#include <iomanip>
#include <time.h>
#include <string>
#include <map>
#include <math.h>
//#include "src/CommandOption.hpp"
//#include "src/CommandOptionWithTimeArg.hpp"
//#include "src/CommandOptionParser.hpp"

#include "gpstk/src/DayTime.hpp"
//#include "src/NovatelStream.hpp"
//#include "src/NovatelData.hpp"
#include "gpstk/src/Exception.hpp"
#include "gpstk/src/EngEphemeris.hpp"
#include "gpstk/src/FFStream.hpp"
#include "gpstk/src/RinexNavData.hpp"
#include "gpstk/src/RinexObsData.hpp"
//#include "src/NovatelStream.hpp"

#include "gpstk/src/RinexObsStream.hpp"
#include "gpstk/src/RinexNavStream.hpp"
#include "gpstk/src/RinexObsHeader.hpp"
#include "gpstk/src/Triple.hpp"

using namespace std;
using namespace gpstk;

using
  boost::any_cast;

class
  NOut:
  public
  NOutUnknown
{
public:
  // Read a log from the internal buffer
  NOutUnknown::RETURN  Read ();
  void  PopItems ();
  // fill header initially
void InitializeHeaders(RinexObsHeader& roh, RinexNavHeader& rnh);
//Fill ObsRinex with data
void RinObsData( RinexObsData& rnod , RANGEB_LOG Log);
//Fill NavRinex with data
void RinNavData( RinexNavData& rnnd , RAWEPHEM_LOG Log);
//Update Header
int UpdateHeader(string& TempFile, string& OutputFile, RinexObsHeader& rh);
//Callculate Utc Time
time_t NOut::UtcTime();
```

```
//Constructor
 NOut();
 //Calculate Rinex File Names
 void RinObsFileCalculate();
 void RinNavFileCalculate();

//////////////////////////////////////GPSTK
  int ndt[9];
double bestdt[9];
// epochs
DayTime CurrEpoch,PrevEpoch,FirstEpoch;
// table of PRN/#obs
map<RinexPrn,vector<int> > table;
map<unsigned long,int> AlmanacAntiSpoof;
vector<int> totals;
// Command line input
bool help,Debug;
DayTime BegTime,EndTime;
string NovatelFile, RinexObsFile, RinexNavFile;
string InputDirectory;
// header fields
bool FillOptionalHeader;
Triple HDAntPos,HDAntOffset;      // TD
vector<string> HDcomments;
vector<RinexObsHeader::RinexObsType> OutputTypes;
long gpsWeek;
bool debias;

bool RinexNavFileFlag;
short ObsFileDay;
short ObsFileYear;
short NavFileDay;
short NavFileYear;
bool AntiSpoof;

int Obsstart_hour;
int Obsstart_min;
int Obsinterval;
int Obscurr_hour;
int Obscurr_min;
int Obsnow;
int Obsstart;
int Obscurr_mod;
int Obsprev_mod;
bool ObsIsClosed;
bool ObsStartInit;
bool IsClosed;

int Navstart_hour;
int Navstart_min;
int Navinterval;
int Navcurr_hour;
int Navcurr_min;
int Navnow;
int Navstart;
int Navcurr_mod;
int Navprev_mod;
bool NavIsClosed;
bool NavStartInit;
```

```
int NavPrevInternval;
int ObsPrevInternval;
string NavFilesPath;
string ObsFilesPath;
string TempFilename;

ofstream myFile;


//-------------------------------------------------------------------------
-----------
// other global data
string ObsFileName;
string NavFileName;

//NovatelStream instr;
RinexObsStream rostr;
RinexNavStream rnstr;
RinexObsHeader roh;        // used in CommandLine
// indexes for the std obs types in the header
int inC1,inP1,inL1,inD1,inS1,inP2,inL2,inD2,inS2;

RinexNavHeader rnh;
RinexNavData rnd;
RinexObsData rod;

/*Ntp Time*/
struct tm ntp_time;
double ntp_secs;

protected:
  void
  DecodeOEM4Binary ();          // Identify Binary log type
   void DecodeOEM4Ascii();           // Identify OEM4 Ascii log type
  Stack <
    boost::any >
    st;

};


#endif
```

```
//       Filename:   LogTypes.h

#ifndef __LOGTYPES_H
#define __LOGTYPES_H

enum BINARY_LOG_TYPE
{
  POSB_LOG_TYPE = 1,
  CLKB_LOG_TYPE = 2,
  TM1B_LOG_TYPE = 3,
  MKTB_LOG_TYPE = 4,
  MKPB_LOG_TYPE = 5,
  SPHB_LOG_TYPE = 6,
  DOPB_LOG_TYPE = 7,
  NAVB_LOG_TYPE = 8,
  DCSB_LOG_TYPE = 9,
  RTCM_LOG_TYPE = 10,
  RNGB_LOG_TYPE = 11,      /* obsolete */
  SATB_LOG_TYPE = 12,
  RCSB_LOG_TYPE = 13,
  REPB_LOG_TYPE = 14,
  RALB_LOG_TYPE = 15,
  IONB_LOG_TYPE = 16,
  UTCB_LOG_TYPE = 17,
  ALMB_LOG_TYPE = 18,
  CTSB_LOG_TYPE = 19,
  EPHB_LOG_TYPE = 20,
  SVPB_LOG_TYPE = 21,
  KPHB_LOG_TYPE = 22,
  RQGB_LOG_TYPE = 24,      /* obsolete */
  CMSB_LOG_TYPE = 25,      /* obsolete */
  PXYB_LOG_TYPE = 26,
  GGAB_LOG_TYPE = 27,
  SVCB_LOG_TYPE = 28,
  CONSOLEDATA_LOG_TYPE = 29,  /* bin logs to support      */
  COM1DATA_LOG_TYPE = 30,  /* pass through data logging  */
  COM2DATA_LOG_TYPE = 31,  /* by P. Fenton Mar 94      */
  RGEB_LOG_TYPE = 32,      /* new rngb that has new cstatus and is smaller
*/
  RGEC_LOG_TYPE = 33,      /* new compressed rngb that has new cstatus and
is smaller */
  VLHB_LOG_TYPE = 34,      /* velocity, latency, heading */
  RT20B_LOG_TYPE = 35,     /* matched obs rt20 (posb) */
  SVDB_LOG_TYPE = 36,      /* another sat ECEF and other related range data
*/
  P20B_LOG_TYPE = 37,
  RTCAB_LOG_TYPE = 38,
  CDSB_LOG_TYPE = 39,      /* new version of CMSB with RTCA */
  MONB_LOG_TYPE = 40,
  RTCM3_LOG_TYPE = 41,
  RTCM9_LOG_TYPE = 42,
  RTCM16_LOG_TYPE = 43,
  RTCM59_LOG_TYPE = 44,
  PVLB_LOG_TYPE = 45,
  DDSB_LOG_TYPE = 46,
  VXYB_LOG_TYPE = 47,
  ETSB_LOG_TYPE = 48,
  PVAB_LOG_TYPE = 49,
  PAVB_LOG_TYPE = 50,
```

```
CLMB_LOG_TYPE = 51,
RBTB_LOG_TYPE = 52,      /* Raw navigation data */
SBTB_LOG_TYPE = 53,      /* strange numbers for compatibility with GSV */
FRMB_LOG_TYPE = 54,
WBRB_LOG_TYPE = 55,
RVSB_LOG_TYPE = 56,
DLLB_LOG_TYPE = 57,
VERB_LOG_TYPE = 58,

BSLB_LOG_TYPE = 59,
RPSB_LOG_TYPE = 60,
RTKB_LOG_TYPE = 61,
RTKOB_LOG_TYPE = 62,
PRTKB_LOG_TYPE = 63,
OPTB_LOG_TYPE = 64,
RGED_LOG_TYPE = 65,      // Same as RGEC with a coded pseudorange SD
RASB_LOG_TYPE = 66,

WRCB_LOG_TYPE = 67,
SNOB_LOG_TYPE = 68,

RTCM1819_LOG_TYPE = 69,
RTCM2021_LOG_TYPE = 70,
RTCM22_LOG_TYPE = 71,
ATTB_LOG_TYPE = 72,
SBLB_LOG_TYPE = 73,
AGCB_LOG_TYPE = 74,      // hidden
ACPB_LOG_TYPE = 75,      // hidden

FRWB_LOG_TYPE = 79,

// LogPak PDC specific logs
PDCB_LOG_TYPE = 76,
PDCDBG1B_LOG_TYPE = 1999,

// WAAS specific
WALB_LOG_TYPE = 81,
WUTCB_LOG_TYPE = 82,
CORB_LOG_TYPE = 83,
MPMB_LOG_TYPE = 95,
CRLB_LOG_TYPE = 96,

SFDB_LOG_TYPE = 98,

// ISM Specific
ISMRB_LOG_TYPE = 124,

MSGB_LOG_TYPE = 1024,
HDRB_LOG_TYPE = 1025,
GRPB_LOG_TYPE = 1026,
DIRB_LOG_TYPE = 1027,
SCHB_LOG_TYPE = 1028,
LPSTATUSB_LOG_TYPE = 1029,
SITELOGB_LOG_TYPE = 1030,
METB_LOG_TYPE = 1031,
BATB_LOG_TYPE = 1032,
PSNB_LOG_TYPE = 1033,
PDCVERB_LOG_TYPE = 1034,
STATUSB_LOG_TYPE = 1035,
```

```
  PROJECTB_LOG_TYPE = 1036,
  GROUPB_LOG_TYPE = 1037,

  DICB_LOG_TYPE = 15400,
  TAGB_LOG_TYPE = 15401,

  // Add new logs here synchronizing numbers with GPSCARD (MINOS1&2) tree


  // leaving the type number for the old RBT, SBT, and FRM binary logs the
same.
  IRBTB_LOG_TYPE = 102,    /* Raw navigation data */
  ISBTB_LOG_TYPE = 103,    /* strange numbers for compatibility with GSV */
  IFRMB_LOG_TYPE = 104,

  // Softsurv format observation blocks
  SSOBS_L1L2_LOG_TYPE = 20001,
  SSOBS_L1_LOG_TYPE = 10001,
  SSOBS_GISMO_LOG_TYPE = 10000,  // now obsolete

  // GLONASS logs
  GEPB_LOG_TYPE = 77,
  GALB_LOG_TYPE = 78,
  CALB_LOG_TYPE = 87,
  GCLB_LOG_TYPE = 88,

  // OEM4 logs
  IONUTCB_LOG_TYPE = 8 | 0x20000,
  CLOCKMODELB_LOG_TYPE = 16 | 0x20000,
  RAWGPSSUBFRAMEB_LOG_TYPE = 25 | 0X20000,
  CHANDEBUGB_LOG_TYPE = 32 | 0x20000,
  VERSIONB_LOG_TYPE = 37 | 0x20000,
  RAWEPHEMB_LOG_TYPE = 41 | 0x20000,
  BESTPOSB_LOG_TYPE = 42 | 0x20000,
  RANGEB_LOG_TYPE = 43 | 0x20000,
  PSRPOSB_LOG_TYPE = 47 | 0x20000,
  SATVISB_LOG_TYPE = 48 | 0x20000,
  PROPAGATEDCLOCKMODELB_LOG_TYPE = 71 | 0x20000,
  ALMANACB_LOG_TYPE = 73 | 0x20000,
  RAWALMB_LOG_TYPE = 74 | 0x20000,
  TRACKSTATB_LOG_TYPE = 83 | 0x20000,
  SATSTATB_LOG_TYPE = 84 | 0x20000,
  RXSTATUSB_LOG_TYPE = 93 | 0x20000,
  RXSTATUSEVENTB_LOG_TYPE = 94 | 0x20000,
  MATCHEDPOSB_LOG_TYPE = 96 | 0x20000,
  BESTVELB_LOG_TYPE = 99 | 0x20000,
  PSRVELB_LOG_TYPE = 100 | 0x20000,
  TIMEB_LOG_TYPE = 101 | 0x20000,
  RANGEPNB_LOG_TYPE = 126 | 0x20000,
  RXCONFIGB_LOG_TYPE = 128 | 0x20000,
  RANGECMPB_LOG_TYPE = 140 | 0x20000,
  RTKPOSB_LOG_TYPE = 141 | 0x20000,
  NAVIGATEB_LOG_TYPE = 161 | 0x20000,
  AVEPOSB_LOG_TYPE = 172 | 0x20000,
  REFSTATIONB_LOG_TYPE = 175 | 0x20000,
  PASSCOM1B_LOG_TYPE = 233 | 0x20000,
  PASSCOM2B_LOG_TYPE = 234 | 0x20000,
  PASSCOM3B_LOG_TYPE = 235 | 0x20000,
```

```
  // OEM4 commands
  FIX_CMD_TYPE = 44 | 0x20000,

  // Zeiss logs supported
  ZMESB_LOG_TYPE = 201,
  ZPOSB_LOG_TYPE = 202,
  ZSTNB_LOG_TYPE = 203,
  ZCFGB_LOG_TYPE = 204,
  ZEPHB_LOG_TYPE = 205,
  ZTAGB_LOG_TYPE = 206,

};
typedef enum BINARY_LOG_TYPE BINARY_LOG_TYPE;
#endif

//       Filename:   NCRC32.hpp
#ifndef __CRC32_HPP
#define __CRC32_HPP

#include "ndefines.h"

class Crc32
{
   private:
      static const unsigned long ulCrcTable[256];   // One copy of the
table is used by all instances of this class

   public:
      unsigned int Test( char *ucBuf_ );                                //
ASCII
      unsigned int Test( unsigned char *ucBuf_, int iSize_ );           //
Binary

      unsigned long CalculateCRC32( unsigned char *ucBuf_, int iSize);  //
Calculate and return the CRC32 for a binary buffer
};

#endif


//       Filename:   NDefines.h
#ifndef __NDEFINES_H
#define __NDEFINES_H

////////////////////////////////////////////////////////////////////////
// Standard Types
////////////////////////////////////////////////////////////////////////
typedef  signed int     INT;     // Signed Machine word size   (Assume to
be no larger than 16 bit)
typedef  unsigned int   UINT;    // Unsigned Machine word size (Assume to
be no larger than 16 bit)

// Defined by microsoft
typedef  char           CHAR;    // Signed 1-byte
typedef  unsigned char  UCHAR;   // Unsigned 1-byte
typedef  signed short   SHORT;   // Signed 2-byte (Unsuported for the T-
805)
typedef  unsigned short USHORT;  // Unsigned 2-byte (Unsuported for the T-
805)
```

```
typedef  signed long    LONG;    // Signed 4-byte
typedef  unsigned long  ULONG;   // Unsigned 4-byte

typedef  float          FLOAT;   // Single precision - IEEE floating point
32 bits
typedef  double         DOUBLE;  // Double precision - IEEE floating point
64 bits
typedef  long double    LDOUBLE; // Extended double precision - 80 bits

typedef  int            BOOL;    // BOOL (TRUE or FALSE)
typedef  unsigned char  BOOLCHAR;// BOOL (TRUE or FALSE) 1-byte


//////////////////////////////////////////////////////////////////////
// Other Global Types
//////////////////////////////////////////////////////////////////////
typedef  DOUBLE         GPS_TIME;   // Time in seconds
typedef  LDOUBLE        BTIME;      // Big Time in bigger seconds

//////////////////////////
// BOOLEAN Values
//////////////////////////
#define TRUE        (1)
#define FALSE       (0)
#define YES         (1)
#define NO          (0)
#define ON          (1)
#define OFF         (0)
#define GOOD        (1)
#define BAD         (0)

//////////////////////////
// NULLs
//////////////////////////
#ifndef  NULL
#define  NULL  (0)    // Pointer
#endif
#ifndef  NUL
#define  NUL   (0)    // End of String
#endif

#define DBL_EQUAL(x,v) ((fabs((DOUBLE) (x) - (DOUBLE) (v) )) < DBL_EPSILON)

// Make character upper case
#define  TOUPPER(ch)    ((ch) >= 'a' && (ch) <= 'z' ? (ch)-'a'+'A' : (ch))

// Convert an ASCII hex character to a nibble
#define  CHARTOHEX(ch)  ((ch) >= 'A' ? ((ch) - 'A')+10 : (ch) - '0')

// 08/12/1999 [CEL]  I added a char in front of the definition for
HEXTOCHAR as
//                   it was generating compiler errors in the form of
//                   significant figures may be lost due to conversion.
// Convert a nibble to a ASCII hex character
#define  HEXTOCHAR(hx)  char((hx) >= 0x0A ? (hx) + 'A' - 0x0A : (hx) + '0')

// Convert a nibble to a ASCII hex character
#define  ISHEX(hx)  (((hx) >= '0' && (hx) <= '9') || (TOUPPER(hx) >= 'A' &&
TOUPPER(hx) <= 'F'))
```

```
// Return the smallest of the two values
#define MIN(x,y)  ((x)<(y) ? (x) : (y))

// Return the largest of the two values
#define MAX(x,y)  ((x)<(y) ? (y) : (x))

// Return the square value
#define SQR(x) ((x)*(x))

// Compare DOUBLES
#define COMPARE_REAL(X, Y, D) ((fabs((X)-(Y)) < (D)) ? 1:0)

// Conversion factors
#define M2INM    ((double) (0.00053995680))  /* INM - International
nautical mile */
#define M2FT     ((double) (1.0/0.3048))
#define M2MILE   ((double) (0.00062137119))
#define MBAR2KPA ((double) (1.0/10.0))
#define MBAR2INI ((double) (1.0/33.8639))
#define MBAR2MM  ((double) (1.0/1.33322))
#define MBAR2PSI ((double) (1.0/68.94757))

// Hex conversion routines; definition ONLY.  Must include "hexconv.cpp" to
use.
unsigned long htol( const char *pszStringIn );
void htoArray( const char *pszStringIn , char* array, int iMaxNum);
void Arraytoh( const char *ArrayIn , char* pszStringOut, int iNumBytes);

#endif

//filename ntpshm.hpp

#ifndef __ntpshm_HPP
#define __ntpshm_HPP
#include <iostream>
#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <math.h>


#define PPS_MAX_OFFSET    100000       /* microseconds the PPS can 'pull'
*/
#define PUT_MAX_OFFSET    500000       /* microseconds for lost lock */

#define NTPD_BASE   0x4e545030   /* "NTP0" */
#define SHM_UNIT    0            /* SHM driver unit number (0..3) */
#define NTPSHMSEGS  4            /* number of NTP SHM segments */

typedef struct  {
    int    mode; /* 0 - if valid set
                *       use values,
                *       clear valid
                * 1 - if valid set
                *       if count before and after read of values is equal,
                *         use values
                *       clear valid
                */
```

```
    int    count;
    time_t clockTimeStampSec;
    int    clockTimeStampUSec;
    time_t receiveTimeStampSec;
    long   receiveTimeStampUSec;
    int    leap;
    int    precision;
    int    nsamples;
    int    valid;
    int    pad[10];
}shmTime;
class NtpShm
{
public:
       shmTime *getShmTime(int unit);
       void ntpshm_init();
       int ntpshm_alloc();
       bool ntpshm_free(int segment);
       int ntpshm_put(double fixtime);
       int ntpshm_pps(struct timeval *tv);
       NtpShm();
       bool enable_ntpshm;
       int shmindex;
       int shmTimeP;
       bool shmTimePPS;
protected:

    shmTime *oshmTime[NTPSHMSEGS];
       bool shmTimeInuse[NTPSHMSEGS];

};
#endif

//filename: ntpshm.cpp

#include "ntpshm.hpp"

using namespace std;
NtpShm::NtpShm()
/*Constructor*/
{
       int i;
       shmindex=-1;
       shmTimeP=-1;
   for(i=0;i<NTPSHMSEGS;i++)
          shmTimeInuse[i]=false;
       shmTimePPS=false;
}
shmTime* NtpShm::getShmTime(int unit)
{
    int shmid=shmget ((key_t)(NTPD_BASE+unit),
                   sizeof (shmTime), IPC_CREAT|0644);
    if (shmid == -1) {
       cout<<"shmget failed\n";
       return NULL;
    } else {
       shmTime *p=( shmTime *)shmat (shmid, 0, 0);
       /*@ -mustfreefresh */
       if ((int)(long)p == -1) {
```

```
            cout<<"shmat failed\n";
            return NULL;
        }
          cout<<"shmat(%d,0,0) succeeded\n";
        return p;
        /*@ +mustfreefresh */
    }
}

void NtpShm::ntpshm_init()
/* attach all NTP SHM segments.  called once at startup, while still root
*/
{
    int i;

    for (i = 0; i < NTPSHMSEGS; i++)
       oshmTime[i] = getShmTime(i);
    memset(shmTimeInuse,0,sizeof(shmTimeInuse));
    shmTimePPS = true;
    enable_ntpshm = true;
}

int NtpShm::ntpshm_alloc()
/* allocate NTP SHM segment.  return its segment number, or -1 */
{
    int i;

    for (i = 0; i < NTPSHMSEGS; i++)
       if (oshmTime[i] != NULL && !shmTimeInuse[i]) {
            shmTimeInuse[i] = true;

            memset((void *)oshmTime[i],0,sizeof(shmTime));
               oshmTime[i]->mode = 1;
            oshmTime[i]->precision = -1; /* initially 0.5 sec */
            oshmTime[i]->nsamples = 3;   /* stages of median filter */
               cout<<"index:"<<i;
            return i;
       }

    return -1;
}

bool NtpShm::ntpshm_free(int segment)
/* free NTP SHM segment */
{
    if (segment < 0 || segment >= NTPSHMSEGS)
       return false;

    shmTimeInuse[segment] = false;
    return true;
}


int NtpShm::ntpshm_put(double fixtime)
/* put a received fix time into shared memory for NTP */
{
    shmTime *myshmTime = NULL;
    struct timeval tv;
    double seconds,microseconds;
```

```
    if (shmindex < 0 ||
       (myshmTime = oshmTime[shmindex]) == NULL)
       return 0;

    (void)gettimeofday(&tv,NULL);
    microseconds = 1000000.0 * modf(fixtime,&seconds);

    myshmTime->count++;
    myshmTime->clockTimeStampSec = (time_t)seconds;
    myshmTime->clockTimeStampUSec = (int)microseconds;
    myshmTime->receiveTimeStampSec = (time_t)tv.tv_sec;
    myshmTime->receiveTimeStampUSec = tv.tv_usec;
    /* setting the precision here does not seem to help anything, too
       hard to calculate properly anyway.  Let ntpd figure it out.
       Any NMEA will be about -1 or -2.
       Garmin GPS-18/USB is around -6 or -7.
    */
    myshmTime->count++;
    myshmTime->valid = 1;

    return 1;
}


/* put NTP shared memory info based on received PPS pulse */

int NtpShm::ntpshm_pps(struct timeval *tv)
{
    shmTime *myshmTime = NULL, *myshmTimeP = NULL;
    time_t seconds;
    double offset;
    long l_offset;

    if (shmindex < 0 || shmTimeP < 0 ||
       (myshmTime = oshmTime[shmindex]) == NULL ||
       (myshmTimeP = oshmTime[shmTimeP]) == NULL)
       return 0;

    /* check if received time messages are within locking range */


    l_offset = myshmTime->receiveTimeStampSec - myshmTime-
>clockTimeStampSec;

    /*@ -ignorequals @*/
    l_offset *= 1000000;
    l_offset += myshmTime->receiveTimeStampUSec - myshmTime-
>clockTimeStampUSec;
    /*@ +ignorequals @*/
    if (labs( l_offset ) > PUT_MAX_OFFSET) {
        cout<<"ntpshm_pps: not in locking range: "<<(long)l_offset<<"\n";
       return -1;
    }
    /*@ -ignorequals @*/
       cout<<"tvusec:"<<tv->tv_usec<<"other:"<<(1000000 - PPS_MAX_OFFSET);
    if (tv->tv_usec < PPS_MAX_OFFSET) {
       seconds = (time_t)tv->tv_sec;
       offset = (double)tv->tv_usec / 1000000.0;
```

```
    } else {
       if (tv->tv_usec > (1000000 - PPS_MAX_OFFSET)) {
           seconds = (time_t)(tv->tv_sec + 1);
           offset = 1 - ((double)tv->tv_usec / 1000000.0);
       }
       if (tv->tv_usec > (1000000 - PUT_MAX_OFFSET)) {
              seconds = (time_t)(tv->tv_sec +1);
              offset = 1 - (double)tv->tv_usec / 1000000.0;
       }else {
           myshmTimeP->precision = -1;  /* lost lock */
           cout<<"ntpshm_pps: lost PPS lock\n";
           return -1;
       }
    }

    myshmTimeP->count++;
    myshmTimeP->clockTimeStampSec = seconds;
    myshmTimeP->clockTimeStampUSec = 0;
    myshmTimeP->receiveTimeStampSec = (time_t)tv->tv_sec;
    myshmTimeP->receiveTimeStampUSec = tv->tv_usec;
    myshmTimeP->precision = offset != 0 ? (int)(ceil(log(offset) / M_LN2))
: -20;
    myshmTimeP->count++;
    myshmTimeP->valid = 1;

    cout<<"ntpshm_pps: clock: "<<(unsigned long)seconds<<" @ "<<(unsigned
long)tv->tv_sec<<"."<<(unsigned long)tv->tv_usec<<", precision
"<<myshmTimeP->precision<<"\n";

    return 1;
}
```

```
//Filename:   NOem4Hdr.h

#ifndef __NOEM4HDR_HPP
#define __NOEM4HDR_HPP

typedef struct OEM4_BINARY_HEADER         // Standard binary header
{
   unsigned char          sop1;          // start of packet first byte
   unsigned char          sop2;          // start of packet second byte
   unsigned char          sop3;          // start of packet third  byte
   unsigned char          header_length; // Length of the header ( From
start of packet )
   unsigned short         number;        // Message number
   unsigned char          type;          // Message type
   unsigned char          port_address;  // Address of the data port the
log was received on
   unsigned short         length;        // Message length (Not including
header or CRC)
   unsigned short         sequence;      // Sequence #
   unsigned char          idle;          // Idle time
   unsigned char          gps_stat;      // GPS Time Status
   unsigned short         gps_week;      // GPS Week number
   unsigned long          millisecs;     // Milliseconds into week
   unsigned long          status;        // Receiver status word
   unsigned short         Reserved;      //
   unsigned short         version;       // Receiver software version
} OEM4_BINARY_HEADER;

enum TIME_STATUS
{
   GPSTIME_UNKNOWN = 0,
   GPSTIME_USER,
   GPSTIME_USERADJUSTING,
   GPSTIME_COARSEADJUSTING,
   GPSTIME_COARSE,
   GPSTIME_COARSESTEERING,
   GPSTIME_FINEADJUSTING,
   GPSTIME_FINE,
   GPSTIME_FINESTEERING,
   GPSTIME_FREEWHEELING,
   GPSTIME_SATTIME,
};

enum PORT_NAME
{
   NO_PORTS = 0,
   COM1_ALL = 1,
   COM2_ALL = 2,
   COM3_ALL = 3,
   USB_ALL = 4,
   THISPORT_ALL = 6,
   ALLPORTS = 8,
   COM1 = 32,
   COM1_1,
   COM1_2,
   COM1_3,
   COM1_4,
   COM1_5,
   COM1_6,
```

```
COM1_7,
COM1_8,
COM1_9,
COM1_10,
COM1_11,
COM1_12,
COM1_13,
COM1_14,
COM1_15,
COM1_16,
COM1_17,
COM1_18,
COM1_19,
COM1_20,
COM1_21,
COM1_22,
COM1_23,
COM1_24,
COM1_25,
COM1_26,
COM1_27,
COM1_28,
COM1_29,
COM1_30,
COM1_31,
COM2 = 64,
COM2_1,
COM2_2,
COM2_3,
COM2_4,
COM2_5,
COM2_6,
COM2_7,
COM2_8,
COM2_9,
COM2_10,
COM2_11,
COM2_12,
COM2_13,
COM2_14,
COM2_15,
COM2_16,
COM2_17,
COM2_18,
COM2_19,
COM2_20,
COM2_21,
COM2_22,
COM2_23,
COM2_24,
COM2_25,
COM2_26,
COM2_27,
COM2_28,
COM2_29,
COM2_30,
COM2_31,
COM3 = 96,
COM3_1,
```

```
COM3_2,
COM3_3,
COM3_4,
COM3_5,
COM3_6,
COM3_7,
COM3_8,
COM3_9,
COM3_10,
COM3_11,
COM3_12,
COM3_13,
COM3_14,
COM3_15,
COM3_16,
COM3_17,
COM3_18,
COM3_19,
COM3_20,
COM3_21,
COM3_22,
COM3_23,
COM3_24,
COM3_25,
COM3_26,
COM3_27,
COM3_28,
COM3_29,
COM3_30,
COM3_31,
USB = 128,
USB_1,
USB_2,
USB_3,
USB_4,
USB_5,
USB_6,
USB_7,
USB_8,
USB_9,
USB_10,
USB_11,
USB_12,
USB_13,
USB_14,
USB_15,
USB_16,
USB_17,
USB_18,
USB_19,
USB_20,
USB_21,
USB_22,
USB_23,
USB_24,
USB_25,
USB_26,
USB_27,
USB_28,
```

```
   USB_29,
   USB_30,
   USB_31,
   THISPORT = 192,
   THISPORT_1,
   THISPORT_2,
   THISPORT_3,
   THISPORT_4,
   THISPORT_5,
   THISPORT_6,
   THISPORT_7,
   THISPORT_8,
   THISPORT_9,
   THISPORT_10,
   THISPORT_11,
   THISPORT_12,
   THISPORT_13,
   THISPORT_14,
   THISPORT_15,
   THISPORT_16,
   THISPORT_17,
   THISPORT_18,
   THISPORT_19,
   THISPORT_20,
   THISPORT_21,
   THISPORT_22,
   THISPORT_23,
   THISPORT_24,
   THISPORT_25,
   THISPORT_26,
   THISPORT_27,
   THISPORT_28,
   THISPORT_29,
   THISPORT_30,
   THISPORT_31,
   MAX_PORT
};

#endif

//filename buffer.cpp
#include "Buffer.hpp"
nBuffer::nBuffer ()
{
  Clear ();               //Zero all the variables
}

void
nBuffer::Clear ()
{
  iBuffStart = 0;
  iBuffEnd = 0;
  iOffset = 0;
  lTotalSize = 0;
  bMyWrapped = FALSE;
}

// Fill the buffer with a string
BUFFER_SET_RETURN nBuffer::Puts (char *szStr_)
```

```
{
  return Set ((unsigned char *) szStr_, strlen (szStr_) + 1);
}

// Fill the buffer with a string
BUFFER_SET_RETURN nBuffer::Set (unsigned char *ucBuf_, int iSize_)
{
  int
    iNew;
  int
    iCallerBuff;          // Up to where the data was read

  // Do we need to shift up?
  if (iOffset != 0)
    {
      if (LONG_MAX - lTotalSize <= iOffset)
       {
         // lTotalSize Wrapped
         bMyWrapped = TRUE;
       }

      if (!bMyWrapped)
       lTotalSize += iOffset;    // Add the size of the buffer just read to
the total size

      iBuffStart -= iOffset;     // Start is where the Get will continue
from

      // Move any data remaining to the start of the private buffer
      for (iNew = 0; iOffset != iBuffEnd;)
       ucBuff[iNew++] = ucBuff[iOffset++];

      iOffset = 0;        // This is the last sync location
      iBuffEnd = iNew;
    }

  if (iBuffEnd + iSize_ > BUFFER_MAX_SIZE * 2)
    return BUFFER_SET_NOT_ENOUGH_ROOM;

  iCallerBuff = 0;        // Reset the offset into the callers buffer

  // Copy the callers data, our internal buffer is twice as big as the max
  while (iSize_--)
    ucBuff[iBuffEnd++] = ucBuf_[iCallerBuff++];
  return BUFFER_SET_GOOD; // All the data has been taken
}

//
*************************************************************************
*
// Read from the buffer
// If ucBuf is NULL no data is output
//
*************************************************************************
BUFFER_GET_RETURN
nBuffer::Get (unsigned char *ucBuf_, int iSize_)
{
  int iCaller;
  int iStart;
```

```
  if (iBuffStart + iSize_ > iBuffEnd)
    return BUFFER_GET_EMPTY;

  iStart = iBuffStart;

  // See if we got it all
  for (iCaller = 0; iSize_; iSize_--)
    {
      // Does the caller want this data or should be skip over it?
      if (ucBuf_)
       {
         // She wants the data
         ucBuf_[iCaller++] = ucBuff[iStart];
       }

      ++iStart;
    }

  iBuffStart = iStart;           // Save for next time in

  return BUFFER_GET_GOOD;
}

//
*************************************************************************
*
 // Give the current offset into ucBuff
 //
*************************************************************************
*
LONG nBuffer::Tell ()
{
  return lTotalSize + (LONG) iBuffStart;
}

    //
*************************************************************************
*
    // Record the offset into ucBuff
    //
*************************************************************************
*
void
nBuffer::SetSync ()
{
  iOffset = iBuffStart;
}

       //
*************************************************************************
*
       // Go back to where we were
       //
*************************************************************************
*
void
nBuffer::Sync ()
{
```

```
  iBuffStart = iOffset;
}

        //
*************************************************************************
*
        // Go to the next character to find data to decode
        //
*************************************************************************
*
void
nBuffer::ReSync (int iSize)
{
  iBuffStart = iOffset + iSize;
}


//  Filename:   NCRC32.cpp

#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "ndefines.h"
#include "ncrc32.hpp"

/*
 *   Crc32::ulCrcTable[256] was generated with the following
 *   standard code:
 *
 *   printf("const unsigned long Crc32::ulCrcTable[256] =\n");
 *   int n, k;
 *   unsigned long c;
 *
 *   printf("{\n");
 *   for (n = 0; n < 256; n++)
 *   {
 *      c = (unsigned long) n;
 *      for (k = 0; k < 8; k++)
 *      {
 *         if (c & 1)
 *            c = 0xedb88320L ^ (c >> 1);
 *         else
 *            c = c >> 1;
 *
 *      }
 *      printf("%s0x%08xL,", n%6 ? " " : "\n   ", c );
 *   }
 *   printf("};\n");
*/
const unsigned long Crc32::ulCrcTable[256] =
{

   0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
0x706af48fL,
   0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL,
0x97d2d988L,
```

```
   0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L, 0x1db71064L,
0x6ab020f2L,
   0xf3b97148L, 0x84be41deL, 0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L,
0x83d385c7L,
   0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL,
0x63066cd9L,
   0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
0xa2677172L,
   0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL, 0x35b5a8faL,
0x42b2986cL,
   0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL,
0xabd13d59L,
   0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L,
0x56b3c423L,
   0xcfba9599L, 0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L,
0xb10be924L,
   0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
0x01db7106L,
   0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L,
0xe8b8d433L,
   0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L, 0x7f6a0dbbL,
0x086d3d2dL,
   0x91646c97L, 0xe6635c01L, 0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L,
0xf262004eL,
   0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L,
0x12b7e950L,
   0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
0xfbd44c65L,
   0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L, 0x4adfa541L,
0x3dd895d7L,
   0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL, 0x346ed9fcL, 0xad678846L,
0xda60b8d0L,
   0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL,
0x270241aaL,
   0xbe0b1010L, 0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L,
0xce61e49fL,
   0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
0x2eb40d81L,
   0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL,
0x74b1d29aL,
   0xead54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L, 0xe3630b12L,
0x94643b84L,
   0x0d6d6a3eL, 0x7a6a5aa8L, 0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L,
0x7d079eb1L,
   0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL,
0x806567cbL,
   0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
0x67dd4accL,
   0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L, 0xd6d6a3e8L,
0xa1d1937eL,
   0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL,
0x48b2364bL,
   0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L, 0xdf60efc3L,
0xa867df55L,
   0x316e8eefL, 0x4669be79L, 0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L,
0x5268e236L,
   0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
0xb2bd0b28L,
```

```
   0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L, 0x2cd99e8bL,
0x5bdeae1dL,
   0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL, 0x9c0906a9L,
0xeb0e363fL,
   0x72076785L, 0x05005713L, 0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL,
0x0cb61b38L,
   0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L,
0xf1d4e242L,
   0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
0x18b74777L,
   0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL, 0x8f659effL,
0xf862ae69L,
   0x616bffd3L, 0x166ccf45L, 0xa00ae278L, 0xd70dd2eeL, 0x4e048354L,
0x3903b3c2L,
   0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL,
0xd9d65adcL,
   0x40df0b66L, 0x37d83bf0L, 0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL,
0x30b5ffe9L,
   0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
0xcdd70693L,
   0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L,
0x2a6f2b94L,
   0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
};

// Calculate and return the CRC32 for a binary buffer
unsigned long
Crc32::CalculateCRC32(
   unsigned char* ucBuf_,
   int            iSize_ )
{
   unsigned long crc = 0;

   for (int n=0; n<iSize_; n++)
      crc = ulCrcTable[(crc ^ ucBuf_[n]) & 0xff] ^ (crc >> 8);

   return crc;
}

// Verify that the given binary log has the correct checksum
unsigned int
Crc32::Test(
   unsigned char* ucBuf_,
   int            iSize_ )
{
   if( ucBuf_[0] != 0xAA )
      FALSE;

   // The crc-32 of a buffer containing a valid crc-32 will be 0.
   return (CalculateCRC32( ucBuf_, iSize_ ) ? FALSE : TRUE);
}

// Verify that the given OEM-4 ascii log has the correct checksum
unsigned int
Crc32::Test(
   char *pcBuf_ )
{
   char*          szCheckCrc32;                // Location of the Crc32
   unsigned long  ulCrc32;                     // Log's Crc32
```

```
   int            dataSize;                     // Length of data used to
compute crc-32
   int            bufSize;                      // Size of buffer including
crc-32
   int            crcSize;
   unsigned char* willy;                       // Temporary buffer
containing binary crc32 appended to guts of pcBuf_
   unsigned int   crcStatus;

   if( pcBuf_[0] != '#' )
      return( FALSE );

   szCheckCrc32 = (char *)strrchr((const char*)pcBuf_, '*');

   if ( !szCheckCrc32 )
      return( FALSE );

   // Convert the ascii crc32 to a binary value
   if( sscanf( szCheckCrc32+1, "%x ", &ulCrc32 ) == 0)
      return( FALSE );

   // Make a copy of the log with the '#' and '*' wrapper removed
   crcSize  = sizeof(ulCrc32);
   dataSize = (int) szCheckCrc32 - (int) &pcBuf_[1];
   bufSize  = dataSize + crcSize;

   willy    = (unsigned char *) malloc(bufSize);
   memset( willy, 0, bufSize);
   memcpy( willy, &pcBuf_[1], dataSize);

   // Append the reported CRC to the data in the buffer
   memcpy(&willy[dataSize], &ulCrc32, crcSize);

   // The crc-32 of a buffer containing a valid crc-32 will be 0.
   crcStatus = CalculateCRC32( willy, bufSize ) ? FALSE : TRUE;

   // ...I had to work too hard for this joke...
   free( willy );

   return crcStatus;
}

//filename GpsOutC.cpp

#include "GpsOutC.hpp"
/*#include "logheaders/range.h"
#include "logheaders/almanac.h"
#include "logheaders/satvis.h"
#include "logheaders/rawephem.h"*/
struct BINARY_HEADER             // Standard binary header
{
  unsigned char
    sop1;                 // start of packet first byte
  unsigned char
    sop2;                 // start of packet second byte
  unsigned char
    sop3;                 // start of packet third  byte
  unsigned char
    crc;                  // the 8 bit CRC check sum
```

```
  unsigned long
    id;                          // Message Id
  unsigned long
    length;               // length of packet
};
NOut::NOut(){

              AntiSpoof=false;
              BegTime = DayTime::BEGINNING_OF_TIME;
              EndTime = DayTime::END_OF_TIME;
              FirstEpoch = DayTime::BEGINNING_OF_TIME;
              ObsIsClosed=true;
              ObsStartInit=true;
              NavIsClosed=true;
              NavStartInit=true;
              IsClosed=true;
              RinexNavFileFlag=false;
              Obsinterval=15;
              Obscurr_mod=-5;
              Navcurr_mod=-5;
              Obsprev_mod=1;
              Obsstart=0;
              Navinterval=60;
              NavPrevInternval=0;
              ObsPrevInternval=0;
              NavFilesPath="/tmp";
              ObsFilesPath="/tmp";
              TempFilename="Obstemp.tmp";


              FirstEpoch = DayTime::BEGINNING_OF_TIME;
              CurrEpoch = DayTime::BEGINNING_OF_TIME;

              myFile.open(("Ascii/ascii.log"),ofstream::out | ofstream::app
);

}

NOutUnknown::RETURN       // Return error status
  NOut::Read ()                  // Read from the internal nBuffer
{
//  nTestXor testxor;           // Check the checksum
  Crc32 testcrc;          // Check the OEM4 crc
//  TESTXOR_RETURN eTestXorResult;      // enum returned from nXor Test
class
  bool eCrcResult;        // enum return from Crc test class
  int NoutReturn;         // Return error status

  // Zero the number of unknown bytes before the next known bytes
  ulLastUnknownBytes = 0L;

  iId = UNKNOWN;
  iType = UNKNOWN_LOG_TYPE;      // Set the grouping to unknown

  for (;;)                // Do this until we find a log
    {
      if (OldNoutReturn == EMPTY)
       {
         SerDat.Sync ();   // Back to where we were
```

```
        OldNoutReturn = OKAY;
       }
      else
       {
        SerDat.SetSync ();      // Save where we are in the serial data
       }

      if (SerDat.Get (&ucBuf[0], 1) == BUFFER_GET_EMPTY)   // Get a
character to identify the log
       {
        return OldNoutReturn = EMPTY;
       }
   switch (ucBuf[0])             // Try to identify the log type
       {
 // Novatel OEM4 ASCII?
         case '#':
         {
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)        // Did we not find a string?
            {
               continue;                     // Can't figure out what the
data is
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            eCrcResult = testcrc.Test((char*)ucBuf);       // Test the
checksum

            // Bad Checksum or wrong function used
            if( eCrcResult != TRUE )
            {
               ReSync();        // We're lost, resync
               continue;
            }

            DecodeOEM4Ascii(); // Try and identify the ascii log

            iNumNOut[iId]++;  // Count it

            m_lTotalSize += m_iSize;

            break;
         }
       case 0xAA:
         {
           // Get the other sync bytes
           if (SerDat.Get (&ucBuf[1], 2) == BUFFER_GET_EMPTY)
             return OldNoutReturn = EMPTY;
           if (ucBuf[1] == 0x44)
             {
              if (ucBuf[2] == 0x12)     // OEM4 binary log
```

```
                  {
                    // Read in the rest of the header
                    if (SerDat.
                       Get (&ucBuf[3],
                            sizeof (OEM4_BINARY_HEADER) - 3) ==
                       BUFFER_GET_EMPTY)
                      return OldNoutReturn = EMPTY;

                    // Is the header size valid?
                    if (((OEM4_BINARY_HEADER *) ucBuf)->header_length !=
                       (unsigned long) sizeof (OEM4_BINARY_HEADER))
                      {
                       ReSync ();   // Mark this as the place to start
                       continue;
                      }

                    // Is the total size reasonable?
                    if (((OEM4_BINARY_HEADER *) ucBuf)->length +
                       (unsigned long) sizeof (OEM4_BINARY_HEADER) >
                       MAX_NOUT_SIZE)
                      {
                       ReSync ();   // Mark this as the place to start
                       continue;
                      }

                    // Coerce the long to an int
                    m_iSize = (int) ((OEM4_BINARY_HEADER *) ucBuf)->length;

                    // Read the rest of the log
                    if (SerDat.Get (ucBuf + sizeof (OEM4_BINARY_HEADER),
                                    m_iSize + OEM4_CRC_SIZE) ==
                       BUFFER_GET_EMPTY)
                      return OldNoutReturn = EMPTY;

                    // We want the size of the whole log
                    m_iSize += sizeof (OEM4_BINARY_HEADER) + OEM4_CRC_SIZE;

                    // Verify the checksum
                    eCrcResult = testcrc.Test (ucBuf, m_iSize);

                    // Bad Checksum or wrong function used or no checksum (not
binary)
                    if (eCrcResult != TRUE)
                      {
                       ReSync ();   // Mark this as the place to start
                       continue;
                      }

                    iType = NOVATEL_OEM4_BINARY; // Assume NovAtel binary

                    DecodeOEM4Binary ();  // Identify log type
                  }
                else
                  {
                    ReSync ();      // Cleanup
                    continue;
                  }
                iNumNOut[iId]++;    // Count it
```

```
             m_lTotalSize += m_iSize;

             break;
            }
         }
         case 'C':
         {
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)        // Did we not find a string?
            {
               continue;                     // Can't figure out what the
data is
            }
            else
            {
               if (NoutReturn == EMPTY)      // There is no more data
               {
                  if (!strncmp((const char*)&ucBuf[1], "om1>", 4))
                  {
                     iType = STANDARD_PROMPT;
                     iId = COM1_PROMPT_LF;
                     m_lTotalSize += m_iSize;
                     break;
                  }
                  else if (!strncmp((const char*)&ucBuf[1], "om2>", 4))
                  {
                     iType = STANDARD_PROMPT;
                     iId = COM2_PROMPT_LF;
                     m_lTotalSize += m_iSize;
                     break;
                  }
                  else
                  {
                     return OldNoutReturn = EMPTY;
                  }
               }
            }

            if (m_iSize == 7 && !strncmp((const char*)&ucBuf[1],
"om1>\r\n", 6))
            {
               iType = STANDARD_PROMPT;
               iId = COM1_PROMPT;
            }
            else if (m_iSize == 6 && !strncmp((const char*)&ucBuf[1],
"om1>\n", 5))
            {
               iType = STANDARD_PROMPT;
               iId = COM1_PROMPT_LF;
            }
            else if (!strncmp((const char*)&ucBuf[1], "om1>", 4))
            {
               iType = STANDARD_PROMPT;
               iId = COM1_PROMPT_LF;
               SerDat.ReSync(5);           // Mark this as the place to
start
               m_iSize = 5;
```

```
            }
            else if (m_iSize == 7 && !strncmp((const char*)&ucBuf[1],
"om2>\r\n", 6))
            {
               iType = STANDARD_PROMPT;
               iId = COM2_PROMPT;
            }
            else if (m_iSize == 6 && !strncmp((const char*)&ucBuf[1],
"om2>\n", 5))
            {
               iType = STANDARD_PROMPT;
               iId = COM2_PROMPT_LF;
            }
            else if (!strncmp((const char*)&ucBuf[1], "om2>", 4))
            {
               iType = STANDARD_PROMPT;
               iId = COM2_PROMPT_LF;
               SerDat.ReSync(5);           // Mark this as the place to
start
               m_iSize = 5;
            }
            else if (m_iSize == 9 && !strncmp((const char*)&ucBuf[1],
"onsole>\n", 8))
            {
               iType = STANDARD_PROMPT;
               iId = CONSOLE_PROMPT;
            }
            else if (m_iSize == 10 && !strncmp((const char*)&ucBuf[1],
"onsole>\r\n", 9))
            {
               iType = STANDARD_PROMPT;
               iId = CONSOLE_PROMPT;
            }
            else
            {
               ReSync();           // Mark this as the place to start
               continue;
            }

            m_lTotalSize += m_iSize;

            break;
         }

         case '\r':
         {
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string
            if (NoutReturn == RESYNC)        // Did we not find a string?
            {
               continue;                     // Can't figure out what the
data is
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
```

```
            }

            if (m_iSize == 2 && !strncmp((const char*)&ucBuf[1], "\n", 1))
            {
               iType = CARD_RESPONSE;
               iId = CRLF_PROMPT;
            }
            else
            {
               ReSync();           // Mark this as the place to start
               continue;
            }

            m_lTotalSize += m_iSize;

            break;
         }

         case '\n':
         {
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)        // Did we not find a string?
            {
               continue;                     // Can't figure out what the
data is
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            if (m_iSize == 3 && !strncmp((const char*)&ucBuf[1], "\r\n",
2))
            {
               iType = CARD_RESPONSE;
               iId = CRLF_PROMPT;
            }
            else
            {
               ReSync();           // Mark this as the place to start
               continue;
            }

            m_lTotalSize += m_iSize;

            break;
         }

         case 'F':
         {
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string
```

```
            if (NoutReturn == RESYNC)        // Copy up to the end of a
string
            {
               continue;
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            if (!strncmp((const char*)&ucBuf[1], "SU Tuned", 8))
            {
               iType = CARD_RESPONSE;
               iId = FSU_TUNED;
            }
            else if (!strncmp((const char*)&ucBuf[1], "SU DeTuned", 10))
            {
               iType = CARD_RESPONSE;
               iId = FSU_DETUNED;
            }
            else
            {
               ReSync();
               continue;
            }

            break;
         }

         case 'I':
         {
            // Invalid Command Name
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)        // Copy up to the end of a
string
            {
               continue;
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            if (!strncmp((const char*)&ucBuf[1], "nvalid Command Name",
19))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_COMMAND_NAME;
               iNumNOut[iId]++;        // Count it
            }
```

```
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Number of
Arguments", 26))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_NUMBER_OF_ARGUMENTS;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Logger
Datatype", 22))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_DATA_LOGGER_TYPE;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Command
Option", 21))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_COMMAND_OPTION;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Channel
Number", 21))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_CHANNEL_NUMBER;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Satellite
Number", 23 ))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_SATELLITE_NUMBER;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Doppler",
14))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_DOPPLER;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid Doppler
Window", 21))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_DOPPLER_WINDOW;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "nvalid", 6))
            {
               iType = CARD_RESPONSE;
               iId = INVALID_IN_DATA;
               iNumNOut[iId]++;        // Count it
            }
            else
            {
               ReSync();
               continue;
```

```
            }

            break;
         }

         case 'N':
         {
            // NVM Error - Unable To Save
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)        // Copy up to the end of a
string
            {
               continue;
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            if (!strncmp((const char*)&ucBuf[1], "VM Error - Unable To
Save", 25))
            {
               iType = CARD_RESPONSE;
               iId = NVM_ERROR;
               iNumNOut[iId]++;        // Count it
            }
            else
            {
               ReSync();
               continue;
            }

            break;
         }

         case '<':
         {
            // Abbreviated ASCII
            NoutReturn = CopyStr(SerDat);       // Copy up to the end of a
string

            if (NoutReturn == RESYNC)
            {
               continue;
            }
            else
            {
               if (NoutReturn == EMPTY)
               {
                  return OldNoutReturn = EMPTY;
               }
            }

            if (!strncmp((const char*)&ucBuf[1], "ERROR:", 6))
```

```
            {
               iType = ERROR_PROMPT;
               iId = INVALID_IN_DATA;
               iNumNOut[iId]++;        // Count it
            }
            else if (!strncmp((const char*)&ucBuf[1], "OK", 2))
            {
               iType = ACK_PROMPT;
               iId = OK_ACK;
               iNumNOut[iId]++;        // Count it
            }
            else
            {
               iType = NOVATEL_OEM4_ABBREVIATED_ASCII;
               iId = UNKNOWN;
               iNumNOut[iId]++;        // Count it
            }

            break;
         }
       default:
         {
           m_lTotalSize++;
           ulUnknownBytes++;
           ulLastUnknownBytes++;
           continue;
         }
       }
       // Is this a lot of unknown bytes?
       if (ulLastUnknownBytes > NOUT_LARGE_UNKNOWN)
        ulMyNumLargeUnknownBytes++;      // Count it

       return (OKAY);
     }
}
////////////////////////////////////////////////////////////////////////////
///////////
void
NOut::DecodeOEM4Ascii()  // member function to identify log, and set iType
and iId appropriately
{
   iId   = UNKNOWN;              // Guess UNKNOWN so we don't need trailing
'else' clauses below
   iType = NOVATEL_OEM4_ASCII;
/*
   // Fill the ncstring with the string up to the comma
   NCString nsLogId((char *)&ucBuf[1], ',');

   // Use the first character to narrow the search.
   switch(ucBuf[1])
   {
      case 'A':
      {
         if (nsLogId == "ALMANACA")
         {
            iId = ALMANACA;
         }
         else if (nsLogId == "AVEPOSA")
         {
```

```
      iId = AVEPOSA;
   }
   break;
}

case 'B':
{
   if (nsLogId == "BESTPOSA")
   {
      iId = BESTPOSA;
   }
   else if (nsLogId == "BESTVELA")
   {
      iId = BESTVELA;
   }
   break;
}

case 'C':
{
   if (nsLogId == "CLOCKMODELA")
   {
      iId = CLOCKMODELA;
   }
   else if (nsLogId == "CHANDEBUGA")
   {
      iId = CHANDEBUGA;
   }
   break;
}

case 'I':
{
   if (nsLogId == "IONUTCA")
   {
      iId = IONUTCA;
   }
   break;
}

case 'M':
{
   if (nsLogId == "MATCHEDPOSA")
   {
      iId = MATCHEDPOSA;
   }
   else if (nsLogId == "MARKPOSA")
   {
      iId = MARKPOSA;
   }
   break;
}

case 'N':
{
   if (nsLogId == "NAVIGATEA")
   {
      iId = NAVIGATEA;
   }
```

```
   break;
}

case 'P':
{
   if (nsLogId == "PSRPOSA")
   {
      iId = PSRPOSA;
   }
   else if (nsLogId == "PSRVELA")
   {
      iId = PSRVELA;
   }
   else if (nsLogId == "PASSCOM1A")
   {
      iId = PASSCOM1A;
   }
   else if (nsLogId == "PASSCOM2A")
   {
      iId = PASSCOM2A;
   }
   else if (nsLogId == "PASSCOM3A")
   {
      iId = PASSCOM3A;
   }
   else if (nsLogId == "PROPAGATEDCLOCKMODELA")
   {
      iId = PROPAGATEDCLOCKMODELA;
   }

   break;
}

case 'R':
{
   if (nsLogId == "RANGEA")
   {
      iId = RANGEA;
   }
   else if (nsLogId == "RANGECMPA")
   {
      iId = RANGECMPA;
   }
   else if (nsLogId == "RAWEPHEMA")
   {
      iId = RAWEPHEMA;
   }
   else if (nsLogId == "RAWGPSSUBFRAMEA")
   {
      iId = RAWGPSSUBFRAMEA;
   }
   else if (nsLogId == "REFSTATIONA")
   {
      iId = REFSTATIONA;
   }
   else if (nsLogId == "RTKPOSA")
   {
      iId = RTKPOSA;
   }
```

```
         else if (nsLogId == "RXCONFIGA")
         {
            iId = RXCONFIGA;
         }
         else if (nsLogId == "RXSTATUSA")
         {
            iId = RXSTATUSA;
         }
         break;
      }

      case 'S':
      {
         if (nsLogId == "SATSTATA")
         {
            iId = SATSTATA;
         }
         break;
      }

      case 'T':
      {
         if (nsLogId == "TIMEA")
         {
            iId = TIMEA;
         }
         else if (nsLogId == "TRACKSTATA")
         {
            iId = TRACKSTATA;
         }
         break;
      }

      case 'V':
      {
         if (nsLogId == "VERSIONA")
         {
            iId = VERSIONA;
         }
         break;
      }

      default:
      {
         break;
      }
   }
*/
   // Unlike OEM3 we will accept OEM4 logs due to the tight CRC

  myFile.write ((char*) ucBuf,m_iSize );
  //cout.write((char*) ucBuf,m_iSize);

   if (iId == UNKNOWN)
   {
      iId = UNKNOWNOEM4A;
   }
}
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////
void
NOut::DecodeOEM4Binary () // Member function to identify the log
{
  ULONG ulOem4Id;         // Converted message id from OEM4 logs

  iType = NOVATEL_OEM4_BINARY;   // Assume NovAtel binary

  iId = UNKNOWN;

  //Calculate the message id
  ulOem4Id = ((OEM4_BINARY_HEADER *) ucBuf)->number;

  // Map the log id into the tlog id
  switch (ulOem4Id | 0x020000L)
    {
    case ALMANACB_LOG_TYPE:
      {
       iId = ALMANACB;
       ALMANACB_LOG myLog;
       myLog = (ALMANACB_LOG &) ucBuf;
       st.push (myLog);
       break;
      }
    case AVEPOSB_LOG_TYPE:
      {
       iId = AVEPOSB;
       break;
      }
    case BESTPOSB_LOG_TYPE:
      {
       iId = BESTPOSB;
       break;
      }
    case BESTVELB_LOG_TYPE:
      {
       iId = BESTVELB;
       break;
      }
    case CLOCKMODELB_LOG_TYPE:
      {
       iId = CLOCKMODELB;
       break;
      }
    case CHANDEBUGB_LOG_TYPE:
      {
       iId = CHANDEBUGB;
       break;
      }
    case IONUTCB_LOG_TYPE:
      {
       iId = IONUTCB;
       break;
      }
    case MATCHEDPOSB_LOG_TYPE:
      {
       iId = MATCHEDPOSB;
       break;
```

```
  }
case NAVIGATEB_LOG_TYPE:
  {
   iId = NAVIGATEB;
   break;
  }
case PASSCOM1B_LOG_TYPE:
  {
   iId = PASSCOM1B;
   break;
  }
case PASSCOM2B_LOG_TYPE:
  {
   iId = PASSCOM2B;
   break;
  }
case PASSCOM3B_LOG_TYPE:
  {
   iId = PASSCOM3B;
   break;
  }
case PSRPOSB_LOG_TYPE:
  {
   iId = PSRPOSB;
   break;
  }
case PSRVELB_LOG_TYPE:
  {
   iId = PSRVELB;
   break;
  }
case PROPAGATEDCLOCKMODELB_LOG_TYPE:
  {
   iId = PROPAGATEDCLOCKMODELB;
   break;
  }
case RANGEB_LOG_TYPE:
  {
   iId = RANGEB;
   RANGEB_LOG myLog;
   myLog = (RANGEB_LOG &) ucBuf;
   st.push (myLog);
   break;
  }
case RANGECMPB_LOG_TYPE:
  {
   iId = RANGECMPB;
   break;
  }
case RAWEPHEMB_LOG_TYPE:
  {
   iId = RAWEPHEMB;
   RAWEPHEM_LOG myLog;
   myLog = (RAWEPHEM_LOG &) ucBuf;
   st.push (myLog);
   break;
  }
case RAWGPSSUBFRAMEB_LOG_TYPE:
  {
```

```
   iId = RAWGPSSUBFRAMEB;
   break;
  }
case REFSTATIONB_LOG_TYPE:
  {
   iId = REFSTATIONB;
   break;
  }
case RTKPOSB_LOG_TYPE:
  {
   iId = RTKPOSB;
   break;
  }
case RXCONFIGB_LOG_TYPE:
  {
   iId = RXCONFIGB;
   break;
  }
case RXSTATUSB_LOG_TYPE:
  {
   iId = RXSTATUSB;
   break;
  }
case SATSTATB_LOG_TYPE:
  {
   iId = SATSTATB;
   break;
  }
case SATVISB_LOG_TYPE:
  {
   //iId = SATVISB;
   SATVISB_LOG myLog;
   myLog = (SATVISB_LOG &) ucBuf;
   st.push (myLog);
   break;
  }
case TIMEB_LOG_TYPE:
  {
   double seconds,microseconds;

   iId = TIMEB;
   TIMEB_LOG myLog;
   myLog = (TIMEB_LOG &) ucBuf;
   ntp_time.tm_year = myLog.body.lUtcYear-1900;
ntp_time.tm_mon = myLog.body.ucUtcMonth-1;
ntp_time.tm_mday = myLog.body.ucUtcDay;
ntp_time.tm_hour = myLog.body.ucUtcHour+2;
ntp_time.tm_min = myLog.body.ucUtcMin;
   ntp_secs = modf(myLog.body.lUtcMillisec/1000,&seconds);
   ntp_time.tm_sec=(int)seconds;
   ntp_time.tm_isdst = -1;
   break;
  }
case TRACKSTATB_LOG_TYPE:
  {
   iId = TRACKSTATB;
   break;
  }
case VERSIONB_LOG_TYPE:
```

```
      {
       iId = VERSIONB;
       break;
      }
    default:
      {
       iId = UNKNOWNOEM4B;
       break;
      }
    }
}
void
NOut::PopItems (){
       int i=0;

         if (!st.empty ()){
                if (st.top().type () == typeid (RANGEB_LOG)){
                     RANGEB_LOG PopLog;
                     PopLog = any_cast < RANGEB_LOG > (st.pop ());//Pop
Range data
                     RinexObsData rod;
                     RinObsData( rod, PopLog);//put data to the gpstk rinex
obs data object

                     if (ObsStartInit){
                            Obsstart = Obsstart_hour * 60 + Obsstart_min;
                            ObsStartInit = false;
                     }
                     Obsprev_mod = Obscurr_mod;
                     Obsnow = Obscurr_hour * 60 + Obscurr_min;
                     if(Obsnow-Obsstart<0)
                            Obsnow+=1440;
                     Obscurr_mod = Obsnow%Obsinterval;
                     cout<<"c:"<<Obscurr_mod<<"-p:"<<Obsprev_mod;
                     if(Obscurr_mod<Obsprev_mod){
                            ObsIsClosed=true;
                     }
                     if( ObsPrevInternval!=Obsinterval && !ObsIsClosed){
                            //rostr.close();
                            ObsIsClosed=true;
                            ObsPrevInternval=Obsinterval;
                     }
                     try{
                     if (ObsIsClosed){
                            InitializeHeaders(roh, rnh);//Initialize header
                            RinObsFileCalculate();    //create Obs Rinex
String File Name
                            ObsFileName = ObsFilesPath + ObsFileName;
                            if(rostr && !IsClosed){
                                   cout<<"Close:"<<ObsFileName;
                                   rostr.close();
                                   UpdateHeader(TempFilename, ObsFileName,
roh);
                            }
                            rostr.open(TempFilename.c_str(), ios::out);
       //open new Obs file
                            rostr.exceptions(ios::failbit);
                            FirstEpoch = rod.time;
                            rostr << roh;//put Rinex Obs Data to the file
```

```
                        if(!IsClosed)
                              ObsStartInit=true;
                        IsClosed=false;
                        ObsIsClosed=false;

                  }

                  rostr << rod;
                  }
                  catch(FFStreamError& e)
                              {
                                    cout <<"Exception:"<<e;
                                    exit(1);
                              }
                  catch(Exception& e)
                        {
                              cout <<"Exception2:"<<e;
                              exit(1);
                        }
            }
            else if (st.top().type () == typeid (RAWEPHEM_LOG)){
                  RAWEPHEM_LOG PopLog;
                  PopLog = any_cast < RAWEPHEM_LOG > (st.pop ());

                  RinexNavData rnd;
                  RinNavData( rnd, PopLog);
                  if (NavStartInit){
                        Navstart=Navstart_hour * 60 + Navstart_min;
                        NavStartInit=false;
                  }
                  Navprev_mod = Navcurr_mod;
                  Navnow = Navcurr_hour * 60 + Navcurr_min;
                  if(Navnow-Navstart<0)
                        Navnow+=1440;
                  Navcurr_mod = Navnow%Navinterval;
                  cout<<"c:"<<Navcurr_mod<<"-p:"<<Navprev_mod;
                  if(Navcurr_mod<Navprev_mod){
                        NavIsClosed=true;
                  }
                  if( NavPrevInternval!=Navinterval && !NavIsClosed){
                        NavIsClosed=true;
                        NavPrevInternval=Navinterval;
                  }
                  try{
                  if (NavIsClosed){
                        InitializeHeaders(roh, rnh);//Initialize header
                        RinNavFileCalculate();    //create Nav Rinex
String File Name
                        NavFileName = NavFilesPath + NavFileName;
                        if(rnstr){
                              cout<<"Close:"<<NavFileName;
                              rnstr.close();

                        }
                        rnstr.open(NavFileName.c_str(), ios::out);
      //open new Nav file
                        rnstr.exceptions(ios::failbit);
```

```
                        rnstr << rnh;
                //put Rinex Nav Data to the file
                        NavIsClosed=false;
                        NavStartInit=true;

                }


                rnstr << rnd;
                }
                catch(FFStreamError& e)
                        {
                                cout <<"Exception:"<<e;
                                exit(1);
                        }
                catch(Exception& e)
                        {
                                cout <<"Exception2:"<<e;
                                exit(1);
                        }


        }
        else if (st.top ().type () == typeid (ALMANACB_LOG)){
                //ALMANACB_LOG PopLog;
                ALMANACB_LOG PopLog;
                PopLog = any_cast < ALMANACB_LOG > (st.pop ());
                if(!AlmanacAntiSpoof.empty())
                        AlmanacAntiSpoof.clear();
                for(i=0;i<PopLog.lNumMessages;i++){
                        AlmanacAntiSpoof.insert(
make_pair(PopLog.data[i].ulPRN,PopLog.data[i].bAntiSpoof) );
                        cout<<"Anti-"<<PopLog.data[i].ulPRN<<"-
"<<PopLog.data[i].bAntiSpoof<<"-\n";
                }
        }
        else if (st.top ().type () == typeid (SATVISB_LOG)){
                  //SATVISB_LOG PopLog;
                  SATVISB_LOG PopLog;
                  PopLog = any_cast < SATVISB_LOG > (st.pop ());
                  printf ("SATVIS - HEADER->>%d,%d,%lu\n",
PopLog.header.bSatVis,
                     PopLog.header.bCompAlm, PopLog.header.ulNumSat);
                    for (int i = 0; i < (unsigned long)
PopLog.header.ulNumSat; i++){
                          printf ("SATVIS - -DATA-
>>%d,%d,%ul,%lf,%lf,%lf,%lf\n",
                                  PopLog.data[i].sPrn,
PopLog.data[i].sReserved,
                                  PopLog.data[i].ulHealth,
PopLog.data[i].dElev,
                                  PopLog.data[i].dAz,
PopLog.data[i].dTrueDop,
                                  PopLog.data[i].dAppDop);
                         //
insdb.insToSatvis(PopLog.data[i].sPrn,PopLog.data[i].sReserved,PopLog.data[
i].ulHealth,PopLog.data[i].dElev,PopLog.data[i].dAz,PopLog.data[i].dTrueDop
,PopLog.data[i].dAppDop);
                }
```

```
            }

        }

}

void NOut::InitializeHeaders(RinexObsHeader& roh, RinexNavHeader& rnh)
{
       DayTime CEpoch;
       CEpoch.setLocalTime();
   // observation header
       if(!OutputTypes.empty())
              OutputTypes.clear();
   roh.version = 2.1;
   roh.fileType = "Observation";
   roh.system = systemGPS;
   roh.fileProgram = "NovaR v1.0";
   roh.fileAgency = "Che";
   roh.date = CEpoch.printf("%04Y/%02m/%02d %02H:%02M:%02S");
   roh.markerName = string(" ");
   roh.markerNumber = string(" ");
   roh.observer = string("George Kirill");
   roh.agency = string("TUC");

   roh.recNo = " ";
   roh.recType = "Novatel";
   roh.recVers = "OEM4"; // defined later by data
   roh.antNo = " ";
   roh.antType = " ";
   //roh.system = systemGPS;

   roh.antennaPosition = Triple(0.0,0.0,0.0);
   roh.antennaOffset = Triple(0.0,0.0,0.0);
   roh.wavelengthFactor[0] = 1;
   roh.wavelengthFactor[1] = 1;

   // must keep track of indexes - for use in table
   if(Debug) cout << "Output obs types and indexes:";
   inC1 = inP1 = inL1 = inD1 = inS1 = inP2 = inL2 = inD2 = inS2 = -1;

       OutputTypes.push_back(RinexObsHeader::C1);
       inC1=0;
       OutputTypes.push_back(RinexObsHeader::D1);
       inD1=1;
       OutputTypes.push_back(RinexObsHeader::D2);
       inD2=2;
       OutputTypes.push_back(RinexObsHeader::L2);
       inL2=3;
       OutputTypes.push_back(RinexObsHeader::L1);
       inL1=4;
       OutputTypes.push_back(RinexObsHeader::P2);
       inP2=5;
    OutputTypes.push_back(RinexObsHeader::S1);
       inS1=6;
    OutputTypes.push_back(RinexObsHeader::S2);
       inS2=7;


   if(Debug) cout << endl;
```

```
   roh.obsTypeList = OutputTypes;

   roh.interval = 10.; // defined later by data
   roh.firstObs = CEpoch; // defined later by data
   roh.firstSystem = systemGPS;
   //roh.firstSystem = 1;
   roh.lastObs = CEpoch; // defined later by data

   roh.valid = RinexObsHeader::allValid21;
   roh.valid |= RinexObsHeader::commentValid;

   // navigation header
   rnh.version = 2.1;
   rnh.fileType = "Observation";
   rnh.fileProgram = roh.fileProgram;
   rnh.date = CEpoch.printf("%04Y/%02m/%02d %02H:%02M:%02S");

   rnh.valid = RinexNavHeader::allValid21;
   rnh.valid |= RinexNavHeader::commentValid;
}
//Observation Data
void NOut::RinObsData( RinexObsData& rnod , RANGEB_LOG Log){
       // all OEM4 obs records
        int i,j;
        int lli;
        int ssi;
      short temps;
      //long nobs;            // number of observation records (may be
2/PRN: L1 and L2)
      RinexPrn sat;
     //RinexObsData rnod;     // this will be returned
      RinexObsData::RinexDatum rd;
      RinexObsData::RinexPrnMap::iterator satit;
      RinexObsData::RinexObsTypeMap::iterator obsit;
        map<unsigned long,int>::iterator itr;
        DayTime FileEpoch;

        /*Times to Calculate FileName*/
            //FileEpoch.setANSI(UtcTime());
                  // put timetag into rnod
         rnod.time =
DayTime(Log.hdr.gps_week,double(Log.hdr.millisecs)/1000.);
         rnod.epochFlag = 0;
         rnod.clockOffset = 0.0;     // don't have it ?

             FileEpoch=rnod.time;
             Obsstart_hour=FileEpoch.hour();
             Obsstart_min=FileEpoch.minute();
             cout<<Obsstart_min<<"@";
             Obscurr_hour=FileEpoch.hour();
             Obscurr_min=FileEpoch.minute();


               /*Information for the header update*/
              PrevEpoch = CurrEpoch;
              CurrEpoch = rnod.time;


            //nobs = 0;
```

```
            //memmove(&nobs, &Log.iObs, 4);
            //intelToHost(nobs);

            rnod.numSvs = 0;
            for(i=0; i<Log.iObs; i++) {
               unsigned short prn,reserved;
               unsigned long TrackStatus;
               float PrStd,PhStd,Doppler,SNR,locktime;
               double Pr,Ph;
               // break out the TrackStatus
               int TrackState = int( Log.data[i].ch_status & 0x0000001FL);
               int Channel    = int((Log.data[i].ch_status & 0x000003E0L)
>>  5);
               bool PhaseLock = bool(Log.data[i].ch_status & 0x00000400L);
               bool CodeLock  = bool(Log.data[i].ch_status & 0x00001000L);
               int Frequency  = int((Log.data[i].ch_status & 0x00600000L)
>> 21); // 0:L1 1:L2
               // CodeType is 0CA 1P 2Pcodeless
               int CodeType   = int((Log.data[i].ch_status & 0x03800000L)
>> 23);
               bool HalfCycle = bool(Log.data[i].ch_status & 0x10000000L);

                       //Check Sats for AntiSpoof
                       itr=AlmanacAntiSpoof.find(Log.data[i].svprn);
                       if(itr!=AlmanacAntiSpoof.end() && itr->second!=0)
                              AntiSpoof=true;
                       else
                              AntiSpoof=false;
                           lli=0;
                           ssi=0;
             //if(!PhaseLock || !CodeLock) continue;        // data is not
reliable
                           //LLI
                           if (!PhaseLock)
                                  lli=lli+1;
                           if (HalfCycle)
                                  lli=lli+2;
                           if (AntiSpoof)
                                  lli=lli+4;
                           //SSI
                           if(Log.data[i].sno < 33)
                                  ssi=1;
                           else if(Log.data[i].sno >= 33 && Log.data[i].sno
< 36)
                                  ssi=2;
                           else if(Log.data[i].sno >= 36 && Log.data[i].sno
< 39)
                                  ssi=3;
                           else if(Log.data[i].sno >= 39 && Log.data[i].sno
< 42)
                                  ssi=4;
                           else if(Log.data[i].sno >= 42 && Log.data[i].sno
< 45)
                                  ssi=5;
                           else if(Log.data[i].sno >= 45 && Log.data[i].sno
< 48)
                                  ssi=6;
                           else if(Log.data[i].sno >= 48 && Log.data[i].sno
< 51)
```

```
                        ssi=7;
                    else if(Log.data[i].sno >= 51 && Log.data[i].sno
< 54)
                        ssi=8;
                    else if(Log.data[i].sno >= 54)
                        ssi=9;


                // fill RinexObsData rnod
                sat = RinexPrn(Log.data[i].svprn,systemGPS);
                satit = rnod.obs.find(sat);          // find the sat
                if(satit == rnod.obs.end()) {        // not there - add this
sat
                   RinexObsData::RinexObsTypeMap rotm;
                   rnod.obs[sat] = rotm;
                   rnod.numSvs++;
                   satit = rnod.obs.find(sat);       // now find it
                }

                RinexObsData::RinexObsTypeMap& obs = satit->second;
                if(Frequency == 0) {       // frequency = L1
                   rd.ssi = ssi;
                            rd.lli = lli; rd.data = -Log.data[i].adr;
                   obs[RinexObsHeader::L1] = rd;                      // L1

                   rd.ssi = ssi;
                            rd.lli = lli; rd.data = Log.data[i].psr;
                   if(CodeType == 0)
                                obs[RinexObsHeader::C1] = rd;    // C1
                   else
                                obs[RinexObsHeader::P1] = rd;    // P1

                   rd.ssi = 0;
                            rd.lli = 0; rd.data = Log.data[i].dop;
                   obs[RinexObsHeader::D1] = rd;                      // D1

                   rd.ssi = 0;
                            rd.lli = 0; rd.data = Log.data[i].sno;
                   obs[RinexObsHeader::S1] = rd;                      // S1

                }
                else {
                   rd.ssi = ssi;
                            rd.lli = lli; rd.data = -Log.data[i].adr;
                   obs[RinexObsHeader::L2] = rd;                      // L2

                   rd.ssi = ssi;
                            rd.lli = lli; rd.data = Log.data[i].psr;
                   obs[RinexObsHeader::P2] = rd;                      // P2

                   rd.ssi = 0;
                            rd.lli = 0; rd.data = Log.data[i].dop;
                   obs[RinexObsHeader::D2] = rd;                      // D2

                   rd.ssi = 0;
                            rd.lli = 0; rd.data = Log.data[i].sno;
                   obs[RinexObsHeader::S2] = rd;                    // S2

                }
```

```
            //} // end RANGE record
              }

}
//Navigation Data
void NOut::RinNavData( RinexNavData& rnnd , RAWEPHEM_LOG Log){
       int i,j,k;
    long templ;
    EngEphemeris eeph;
      DayTime FileEpoch;

             try{

             //Navigation Data Follow
           // parse data
         short prn,track=1;
         long gpsSOW;

           // get PRN and timetag
         prn = short(Log.data.prn);
           // convert the 3 subframes and create EngEphemeris
            long subframe[10];

             k=0;
         for(i=0; i<10; i++){
                    subframe[i] = (Log.data.subframe1[k] <<
22)+(Log.data.subframe1[k+1] << 14)+(Log.data.subframe1[k+2] << 6);
            k += 3;
             }

if(!eeph.addSubframe(subframe,Log.data.ereferweek,prn,track)){
              cout << "Failed to convert RAWEPH subframe " << j+1 << ",
prn " << prn
                 << " at time " << Log.data.ereferweek << " " <<
Log.data.erefertime << endl;
            }

            k=0;
            for(i=0; i<10; i++){
                   subframe[i] = (Log.data.subframe2[k] <<
22)+(Log.data.subframe2[k+1] << 14)+(Log.data.subframe2[k+2] << 6);
            k += 3;
            }
            if(!eeph.addSubframe(subframe,Log.data.ereferweek,prn,track)){
              cout << "Failed to convert RAWEPH subframe " << j+1 << ",
prn " << prn
                 << " at time " << Log.data.ereferweek << " " <<
Log.data.erefertime << endl;
            }

            k=0;
            for(i=0; i<10; i++){
                   subframe[i] = (Log.data.subframe3[k] <<
22)+(Log.data.subframe3[k+1] << 14)+(Log.data.subframe3[k+2] << 6);
            k += 3;
        }
             if(!eeph.addSubframe(subframe,Log.data.ereferweek,prn,track)){
```

```
               cout << "Failed to convert RAWEPH subframe " << j+1 << ",
prn " << prn
                  << " at time " << Log.data.ereferweek << " " <<
Log.data.erefertime << endl;
             }
             cout<<"NavWeek: "<<Log.data.ereferweek<<"*\n";

             rnnd=gpstk::RinexNavData(eeph);
             //FileEpoch.setANSI(UtcTime());
             FileEpoch=rnnd.time;
              NavFileDay=FileEpoch.DOYday();
              NavFileYear=FileEpoch.DOYyear();
              Navstart_hour=FileEpoch.hour();
              Navstart_min=FileEpoch.minute();
              cout<<Navstart_min<<"@";
              Navcurr_hour=FileEpoch.hour();
              Navcurr_min=FileEpoch.minute();
              }
              catch(gpstk::FFStreamError& e){
                     cout << e;
                     exit(1);
                     }
             catch(gpstk::Exception& e){
                    cout << e;
                    exit(1);
                    }

}
void NOut::RinObsFileCalculate(){
       //Calculate the Name of the new Rinex Observation File
       stringstream ss;
       string station;
       string DOY;
       string f("abcdefghijklmnopqrstuvwxyz0");
       string year;
       string minute;
       string FileType;

       station="tuc1";
       ss<<FirstEpoch.DOYday();
       DOY=ss.str();
       ss.str("");
       ss<<FirstEpoch.DOYyear();
       year=ss.str();
       FileType="o";
       ss.str("");
       ss<<FirstEpoch.minute();
       minute=ss.str();
       if(FirstEpoch.minute()<10)
              minute="0" + minute;
       if (Obsinterval>=60 && Obsinterval<1440)
              ObsFileName=station + DOY + f[FirstEpoch.hour()] + "." +
year.substr(2) + FileType;
       else if(Obsinterval==1440)
              ObsFileName=station + DOY + "0" + "." + year.substr(2) +
FileType;
       else
              ObsFileName=station + DOY + f[FirstEpoch.hour()] + minute +
"." + year.substr(2) + FileType;
```

```
}
void NOut::RinNavFileCalculate(){
       //Calculate the Name of the new Rinex Navigation File
       stringstream ss;
       string station;
       string DOY;
       string f("abcdefghijklmnopqrstuvwxyz");
       string year;
       string minute;
       string FileType;

       station="tuc1";
       ss<<NavFileDay;
       DOY=ss.str();
       ss.str("");
       ss<<NavFileYear;
       year=ss.str();
       FileType="n";
       ss.str("");
       ss<<Navstart_min;
       minute=ss.str();
       if(Navstart_min<10)
              minute="0" + minute;
       if (Navinterval>=60 && Navinterval<1440){
              NavFileName=station + DOY + f[Navstart_hour] + "." +
year.substr(2) + FileType;
       }
       else if(Navinterval==1440)
              NavFileName=station + DOY + "0" + "." + year.substr(2) +
FileType;
       else
              NavFileName=station + DOY + f[Navstart_hour] + minute + "." +
year.substr(2) + FileType;
}
int NOut::UpdateHeader(string& TempFile, string& OutputFile,
RinexObsHeader& rh){
       rh.firstObs = FirstEpoch;
    rh.lastObs = PrevEpoch;

       // re-open the file and replace the header
   RinexObsHeader rhjunk;
   RinexObsStream InAgain(TempFile.c_str());
   RinexObsStream ROutStr(OutputFile.c_str(), ios::out);
   InAgain.exceptions(fstream::failbit);
   ROutStr.exceptions(fstream::failbit);

   InAgain >> rhjunk;
   ROutStr << rh;

   RinexObsData robs;
   while(InAgain >> robs)
      ROutStr << robs;

   //InAgain.clear();
   InAgain.close();
   //ROutStr.clear();
   ROutStr.close();

      // delete the temporary
```

```
       /*if(remove(TempFile.c_str()) != 0) {
      cerr << "Error: Could not remove existing temp file: " << TempFile <<
endl;
       return -1;
   }*/
   //else if(Debug) cout << "Deleted temporary file " << TempFile << endl;

   return 0;
}

time_t NOut::UtcTime(){
       time_t curr=time(0);// current local time
       tm local=*gmtime(&curr);// convert curr to GMT, store as tm
       time_t utc=(mktime(&local));// convert GMT tm to GMT time_t
       return utc;
}

//filename GpsOut.cpp

#include "GpsOut.hpp"
//#include "nxor.cpp"

struct NLogTypeRecord
{
  NOUT_ID eID;                    // The ID as specified in NOutUnknown.hpp.
  char *szName;                   // The ASCII version of the ID.
  char *szDesc;                   // Brief description of the log type.
};

static const NLogTypeRecord g_astLogMap[] = {

  {UNKNOWN, "UNKNOWN", "This identifier is for unknown logs"},
  {ACPA, "ACPA", "Description not available"},
  {AGCA, "AGCA", "Description not available"},
  {ALMA, "ALMA", "Description not available"},
  {ATTA, "ATTA", "Description not available"},
  {BATA, "BATA", "Description not available"},
  {BSLA, "BSLA", "Description not available"},
  {CALA, "CALA", "Description not available"},
  {CDSA, "CDSA", "Description not available"},
  {CLKA, "CLKA", "Description not available"},
  {CLMA, "CLMA", "Description not available"},
  {COM1A, "COM1A", "Description not available"},
  {COM2A, "COM2A", "Description not available"},
  {CONSOLEA, "CONSOLEA", "Description not available"},
  {CORA, "CORA", "Description not available"},
  {CTSA, "CTSA", "Description not available"},
  {DCSA, "DCSA", "Description not available"},
  {DIRA, "DIRA", "Description not available"},
  {DOPA, "DOPA", "Description not available"},
  {ETSA, "ETSA", "Description not available"},
  {FRMA, "FRMA", "Description not available"},
  {FRWA, "FRWA", "Description not available"},
  {GALA, "GALA", "Description not available"},
  {GCLA, "GCLA", "Description not available"},
  {GEPA, "GEPA", "Description not available"},
  {GROUPA, "GROUPA", "Description not available"},
  {GRPA, "GRPA", "Description not available"},
  {HDRA, "HDRA", "Description not available"},
```

```
{IONA, "IONA", "Description not available"},
{ISMRA, "ISMRA", "Description not available"},
{KPHA, "KPHA", "Description not available"},
{LPSTATUSA, "LPSTATUSA", "Description not available"},
{META, "META", "Description not available"},
{MKPA, "MKPA", "Description not available"},
{MKTA, "MKTA", "Description not available"},
{MPMA, "MPMA", "Description not available"},
{MSGA, "MSGA", "Description not available"},
{NAVA, "NAVA", "Description not available"},
{OPTA, "OPTA", "Description not available"},
{P20A, "P20A", "Description not available"},
{PAVA, "PAVA", "Description not available"},
{PDCA, "PDCA", "Description not available"},
{PDCDBG1A, "PDCDBG1A", "Description not available"},
{PDCVERA, "PDCVERA", "Description not available"},
{PNTA, "PNTA", "Description not available"},
{POSA, "POSA", "Description not available"},
{PROJECTA, "PROJECTA", "Description not available"},
{PRTKA, "PRTKA", "Description not available"},
{PSNA, "PSNA", "Description not available"},
{PXYA, "PXYA", "Description not available"},
{PVAA, "PVAA", "Description not available"},
{RALA, "RALA", "Description not available"},
{RASA, "RASA", "Raw Almanac Set ASCII log"},
{RBTA, "RBTA", "Description not available"},
{RCCA, "RCCA", "Description not available"},
{RCSA, "RCSA", "Description not available"},
{REPA, "REPA", "Description not available"},
{RGEA, "RGEA", "Description not available"},
{RNGA, "RNGA", "Description not available"},
{RPSA, "RPSA", "Description not available"},
{RT20A, "RT20A", "Description not available"},
{RTCA, "RTCA", "Description not available"},
{RTCMA, "RTCMA", "Description not available"},
{RTCM16T, "RTCM16T", "Description not available"},
{RTKA, "RTKA", "Description not available"},
{RTKOA, "RTKOA", "Description not available"},
{RVSA, "RVSA", "Description not available"},
{SATA, "SATA", "Description not available"},
{SBLA, "SBLA", "Description not available"},
{SBTA, "SBTA", "Description not available"},
{SCHA, "SCHA", "Description not available"},
{SFDA, "SFDA", "Satellite Failure Detection"},
{SITELOGA, "SITELOGA", "Description not available"},
{SNOA, "SNOA", "Description not available"},
{SPHA, "SPHA", "Description not available"},
{STATUSA, "STATUSA", "Description not available"},
{SVDA, "SVDA", "Description not available"},
{TM1A, "TM1A", "Description not available"},
{UTCA, "UTCA", "Description not available"},
{VERA, "VERA", "Description not available"},
{VLHA, "VLHA", "Description not available"},
{WALA, "WALA", "Description not available"},
{WUTCA, "WUTCA", "Description not available"},
{WBRA, "WBRA", "Description not available"},
{WRCA, "WRCA", "Description not available"},
{ACPB, "ACPB", "Description not available"},
{AGCB, "AGCB", "Description not available"},
```

```
{ALMB, "ALMB", "Description not available"},
{ATTB, "ATTB", "Description not available"},
{BATB, "BATB", "Description not available"},
{BSLB, "BSLB", "Description not available"},
{CALB, "CALB", "Description not available"},
{CDSB, "CDSB", "Description not available"},
{CLKB, "CLKB", "Description not available"},
{CLMB, "CLMB", "Description not available"},
{COM1B, "COM1B", "Description not available"},
{COM2B, "COM2B", "Description not available"},
{CONSOLEB, "CONSOLEB", "Description not available"},
{CORB, "CORB", "Description not available"},
{CTSB, "CTSB", "Description not available"},
{DCSB, "DCSB", "Description not available"},
{DIRB, "DIRB", "Description not available"},
{DLLB, "DLLB", "Description not available"},
{DOPB, "DOPB", "Description not available"},
{ETSB, "ETSB", "Description not available"},
{FRMB, "FRMB", "Description not available"},
{FRWB, "FRWB", "Description not available"},
{GALB, "GALB", "Description not available"},
{GCLB, "GCLB", "Description not available"},
{GEPB, "GEPB", "Description not available"},
{GROUPB, "GROUPB", "Description not available"},
{GRPB, "GRPB", "Description not available"},
{HDRB, "HDRB", "Description not available"},
{IONB, "IONB", "Description not available"},
{ISMRB, "ISMRB", "Description not available"},
{KPHB, "KPHB", "Description not available"},
{LPSTATUSB, "LPSTATUSB", "Description not available"},
{METB, "METB", "Description not available"},
{MKPB, "MKPB", "Description not available"},
{MKTB, "MKTB", "Description not available"},
{MPMB, "MPMB", "Description not available"},
{MSGB, "MSGB", "Description not available"},
{NAVB, "NAVB", "Description not available"},
{OPTB, "OPTB", "Description not available"},
{P20B, "P20B", "Description not available"},
{PAVB, "PAVB", "Description not available"},
{::PDCB, "PDCB", "Description not available"},
{PDCDBG1B, "PDCDBG1B", "Description not available"},
{PDCVERB, "PDCVERB", "Description not available"},
{POSB, "POSB", "Description not available"},
{PROJECTB, "PROJECTB", "Description not available"},
{PRTKB, "PRTKB", "Description not available"},
{PSNB, "PSNB", "Description not available"},
{PVAB, "PVAB", "Description not available"},
{PXYB, "PXYB", "Description not available"},
{RALB, "RALB", "Description not available"},
{RASB, "RASB", "Raw Almanac Set Binary log"},
{RBTB, "RBTB", "Description not available"},
{RCSB, "RCSB", "Description not available"},
{REPB, "REPB", "Description not available"},
{RGEB, "RGEB", "Description not available"},
{RGEC, "RGEC", "Description not available"},
{RGED, "RGED", "Description not available"},
{RPSB, "RPSB", "Description not available"},
{RT20B, "RT20B", "Description not available"},
{RTCAB, "RTCAB", "Description not available"},
```

```
{RTCMB, "RTCMB", "Description not available"},
{RTKB, "RTKB", "Description not available"},
{RTKOB, "RTKOB", "Description not available"},
{RVSB, "RVSB", "Description not available"},
{SATB, "SATB", "Description not available"},
{SBLB, "SBLB", "Description not available"},
{SBTB, "SBTB", "Description not available"},
{SCHB, "SCHB", "Description not available"},
{SFDB, "SFDB", "Satellite Failure Detection"},
{SITELOGB, "SITELOGB", "Description not available"},
{SNOB, "SNOB", "Description not available"},
{SPHB, "SPHB", "Description not available"},
{STATUSB, "STATUSB", "Description not available"},
{SVDB, "SVDB", "Description not available"},
{TM1B, "TM1B", "Description not available"},
{UTCB, "UTCB", "Description not available"},
{VERB, "VERB", "Description not available"},
{VLHB, "VLHB", "Description not available"},
{WALB, "WALB", "Description not available"},
{WUTCB, "WUTCB", "Description not available"},
{WBRB, "WBRB", "Description not available"},
{WRCB, "WRCB", "Description not available"},
{SSOBSL1L2, "SSOBSL1L2", "Description not available"},
{SSOBSL1, "SSOBSL1", "Description not available"},
{SSOBSGISMO, "SSOBSGISMO", "Description not available"},
{TAGB, "TAGB", "Description not available"},
{DICB, "DICB", "Description not available"},
{GPALM, "GPALM", "Description not available"},
{GPGGA, "GPGGA", "Description not available"},
{GPGLL, "GPGLL", "Description not available"},
{GPGNS, "GPGNS", "Description not available"},
{GPGRS, "GPGRS", "Description not available"},
{GPGSA, "GPGSA", "Description not available"},
{GPGST, "GPGST", "Description not available"},
{GPGSV, "GPGSV", "Description not available"},
{GPRMB, "GPRMB", "Description not available"},
{GPRMC, "GPRMC", "Description not available"},
{GPVTG, "GPVTG", "Description not available"},
{GPZDA, "GPZDA", "Description not available"},
{GPZTG, "GPZTG", "Description not available"},
{GLALM, "GLALM", "Description not available"},
{GLGGA, "GLGGA", "Description not available"},
{GLGLL, "GLGLL", "Description not available"},
{GLGNS, "GLGNS", "Description not available"},
{GLGRS, "GLGRS", "Description not available"},
{GLGSA, "GLGSA", "Description not available"},
{GLGST, "GLGST", "Description not available"},
{GLGSV, "GLGSV", "Description not available"},
{GLRMB, "GLRMB", "Description not available"},
{GLRMC, "GLRMC", "Description not available"},
{GLVTG, "GLVTG", "Description not available"},
{GLZDA, "GLZDA", "Description not available"},
{GLZTG, "GLZTG", "Description not available"},
{GNALM, "GNALM", "Description not available"},
{GNGGA, "GNGGA", "Description not available"},
{GNGLL, "GNGLL", "Description not available"},
{GNGNS, "GNGNS", "Description not available"},
{GNGRS, "GNGRS", "Description not available"},
{GNGSA, "GNGSA", "Description not available"},
```

```
  {GNGST, "GNGST", "Description not available"},
  {GNGSV, "GNGSV", "Description not available"},
  {GNRMB, "GNRMB", "Description not available"},
  {GNRMC, "GNRMC", "Description not available"},
  {GNVTG, "GNVTG", "Description not available"},
  {GNZDA, "GNZDA", "Description not available"},
  {GNZTG, "GNZTG", "Description not available"},
  // Timble NMEA.
  {PTNL_GGK, "PTNL_GGK", "Description not available"},
  {XOBS, "XOBS", "Description not available"},
  {XOHD, "XOHD", "Description not available"},
  {XNHD, "XNHD", "Description not available"},
  {XNAV, "XNAV", "Description not available"},
  {ZMESB, "ZMESB", "Description not available"},
  {ZPOSB, "ZPOSB", "Description not available"},
  {ZEPHB, "ZEPHB", "Description not available"},
  {ZSTNB, "ZSTNB", "Description not available"},
  {ZCFGB, "ZCFGB", "Description not available"},
  {ZTAGB, "ZTAGB", "Description not available"},
  {TESTBED_TIMEA, "TESTBED_TIMEA", "Description not available"},
  {TESTBED_TESTIDA, "TESTBED_TESTIDA", "Description not available"},
  {TESTBED_REPORTA, "TESTBED_REPORTA", "Description not available"},
  {TESTBED_MIRRORA, "TESTBED_MIRRORA", "Description not available"},
  {TESTBED_TIMEB, "TESTBED_TIMEB", "Description not available"},
  {TESTBED_TESTIDB, "TESTBED_TESTIDB", "Description not available"},
  {TESTBED_REPORTB, "TESTBED_REPORTB", "Description not available"},
  {TESTBED_MIRRORB, "TESTBED_MIRRORB", "Description not available"},
  // Card response category.
  {FSU_TUNED, "FSU_TUNED", "Description not available"},
  {FSU_DETUNED, "FSU_DETUNED", "Description not available"},
  {TRANSPARENTMODE_ON, "TRANSPARENTMODE_ON", "TRANSPARENTMODE ON command"},
  {TRANSPARENTMODE_OFF, "TRANSPARENTMODE_OFF", "TRANSPARENTMODE ON
command"},
  // NovAtel propmt category.
  {COM1_PROMPT, "COM1_PROMPT", "Receiver COM1 prompt"},
  {COM2_PROMPT, "COM2_PROMPT", "Receiver COM2 prompt"},
  {COM3_PROMPT, "COM3_PROMPT", "Receiver COM3 prompt"},
  // Some prompts end in only a line feed.
  {COM1_PROMPT_LF, "COM1_PROMPT_LF", "Description not available"},
  {COM2_PROMPT_LF, "COM2_PROMPT_LF", "Description not available"},
  {COM3_PROMPT_LF, "COM3_PROMPT_LF", "Description not available"},
  {CONSOLE_PROMPT, "CONSOLE_PROMPT", "Description not available"},
  // NovAtel likes to spit out extra CR/LFs every now and then.
  {CRLF_PROMPT, "CRLF_PROMPT", "Description not available"},
  {INVALID_COMMAND_NAME, "INVALID_COMMAND_NAME",
   "An invalid command was entered"},
  {INVALID_NUMBER_OF_ARGUMENTS, "INVALID_NUMBER_OF_ARGUMENTS",
   "The command had an invalid number of arguments"},
  {INVALID_DATA_LOGGER_TYPE, "INVALID_DATA_LOGGER_TYPE",
   "An invalid log was requested"},
  {INVALID_COMMAND_OPTION, "INVALID_COMMAND_OPTION",
   "A command was used incorrectly"},
  {INVALID_IN_DATA, "INVALID_IN_DATA",
   "Some sort of \"Invalid\" message was received"},
  {INVALID_CHANNEL_NUMBER, "INVALID_CHANNEL_NUMBER",
   "The specified channel number was invalid"},
  {INVALID_SATELLITE_NUMBER, "INVALID_SATELLITE_NUMBER",
   "The specified satellite number was invalid"},
```

```
  {INVALID_DOPPLER, "INVALID_DOPPLER", "The specified doppler was
invalid"},
  {INVALID_DOPPLER_WINDOW, "INVALID_DOPPLER_WINDOW",
   "The specified doppler window was invalid"},
  {NVM_ERROR, "NVM_ERROR", "NVM Error - Unable To Save"},
  {OK_ACK, "OK_ACK", "OK Acknowledge - Command entry okay"},
  // OEM4
  {CLOCKMODELA, "CLOCKMODELA", "OEM4 clock model data"},
  {CLOCKMODELB, "CLOCKMODELB", "OEM4 clock model data"},
  {VERSIONA, "VERSIONA", "OEM4 version information"},
  {VERSIONB, "VERSIONB", "OEM4 version information"},
  {AVEPOSA, "AVEPOSA", "Position Averaging"},
  {AVEPOSB, "AVEPOSB", "Position Averaging"},
  {BESTPOSA, "BESTPOSA", "Best OEM4 position available"},
  {BESTPOSB, "BESTPOSB", "Best OEM4 position available"},
  {BESTVELA, "BESTVELA", "Best OEM4 velocity available"},
  {BESTVELB, "BESTVELB", "Best OEM4 velocity available"},
  {MARKPOSA, "MARKPOSA", "OEM4 Marked Position Log"},
  {MARKPOSB, "MARKPOSB", "OEM4 Marked Position Log"},
  {MATCHEDPOSA, "MATCHEDPOSA", "Time matched OEM4 position log"},
  {MATCHEDPOSB, "MATCHEDPOSB", "Time matched OEM4 position log"},
  {NAVIGATEA, "NAVIGATEA", "User Navigation Data"},
  {NAVIGATEB, "NAVIGATEB", "User Navigation Data"},
  {PASSCOM1A, "PASSCOM1A", "OEM4 Pass-Through Log"},
  {PASSCOM2A, "PASSCOM2A", "OEM4 Pass-Through Log"},
  {PASSCOM3A, "PASSCOM3A", "OEM4 Pass-Through Log"},
  {PASSCOM1B, "PASSCOM1B", "OEM4 Pass-Through Log"},
  {PASSCOM2B, "PASSCOM2B", "OEM4 Pass-Through Log"},
  {PASSCOM3B, "PASSCOM3B", "OEM4 Pass-Through Log"},
  {PSRPOSA, "PSRPOSA", "OEM4 Pseudorange Position"},
  {PSRPOSB, "PSRPOSB", "OEM4 Pseudorange Position"},
  {PSRVELA, "PSRVELA", "OEM4 Pseudorange Velocity"},
  {PSRVELB, "PSRVELB", "OEM4 Pseudorange Velocity"},
  {PROPAGATEDCLOCKMODELA, "PROPAGATEDCLOCKMODELA",
   "OEM4 Propagated Range Variance & Bias"},
  {PROPAGATEDCLOCKMODELB, "PROPAGATEDCLOCKMODELB",
   "OEM4 Propagated Range Variance & Bias"},
  {RTKPOSA, "RTKPOSA", "Short OEM4 position log"},
  {RTKPOSB, "RTKPOSB", "Short OEM4 position log"},
  {RANGEA, "RANGEA", "OEM4 range log"},
  {RANGEB, "RANGEB", "OEM4 range log"},
  {RANGECMPA, "RANGECMPA", "OEM4 compressed range log"},
  {RANGECMPB, "RANGECMPB", "OEM4 compressed range log"},
  {RAWEPHEMA, "RAWEPHEMA", "OEM4 raw ephemeris data"},
  {RAWEPHEMB, "RAWEPHEMB", "OEM4 raw ephemeris data"},
  {RAWGPSSUBFRAMEA, "RAWGPSSUBFRAMEA", "Description not available"},
  {RAWGPSSUBFRAMEB, "RAWGPSSUBFRAMEB", "Description not available"},
  {REFSTATIONA, "REFSTATIONA", "OEM4 Reference Station Position and
Health"},
  {REFSTATIONB, "REFSTATIONB", "OEM4 Reference Station Position and
Health"},
  {RXCONFIGA, "RXCONFIGA", "OEM4 receiver configuration information"},
  {RXCONFIGB, "RXCONFIGB", "OEM4 receiver configuration information"},
  {RXSTATUSA, "RXSTATUSA", "OEM4 Receiver Status log"},
  {RXSTATUSB, "RXSTATUSB", "OEM4 Receiver Status log"},
  {SATSTATA, "SATSTATA", "Description not available"},
  {SATSTATB, "SATSTATB", "Description not available"},
  {TIMEA, "TIMEA", "OEM4 Time Data"},
  {TIMEB, "TIMEB", "OEM4 Time Data"},
```

```
  {TRACKSTATA, "TRACKSTATA", "OEM4 Tracking Status log"},
  {TRACKSTATB, "TRACKSTATB", "OEM4 Tracking Status log"},
  {ALMANACA, "ALMANACA", "OEM4 Almanac data"},
  {ALMANACB, "ALMANACB", "OEM4 Almanac data"},
  {IONUTCA, "IONUTCA", "Description not available"},
  {IONUTCB, "IONUTCB", "Description not available"},
  {CHANDEBUGA, "CHANDEBUGA", "OEM4 debug log"},
  {CHANDEBUGB, "CHANDEBUGB", "OEM4 debug log"},
  {UNKNOWNOEM4A, "UNKNOWNOEM4A", "Unknown OEM4 log"},
  {UNKNOWNOEM4B, "UNKNOWNOEM4B", "Unknown OEM4 log"},
  {POINTOBS, "POINTOBS", "Point NCObservation"},
  {POINTEPH, "POINTEPH", "Point NCOrbit"},
  // Ashtech PASHR,POS log.
  {PASHR_POS, "PASHR_POS", "Description not available"},
  {LOGS_START, "NA", "Simply an ENUM marker"},
  {LOGS_END, "NA", "Simply an ENUM marker"},
  {MAX_NOUT_ID, "MAX_NOUT_ID", "Description not available"},
  {ALL_NOUT_ID, "ALL_NOUT_ID", "Description not available"},
};

NOutUnknown::NOutUnknown ()
{
  m_ppszId = NULL;        // This table only has to be built once forever
  Clear ();               // Clear everything
}

NOutUnknown::~NOutUnknown ()
{
  if (m_ppszId != NULL)
    {
      delete m_ppszId;
      m_ppszId = NULL;
    }
}

void
NOutUnknown::Clear ()
{
  memset (iNumNOut, 0, sizeof (INT) * MAX_NOUT_ID);  // zero the log counter
  iId = UNKNOWN;          // zero the last log ID
  iType = UNKNOWN_LOG_TYPE;      // zero the last log type
  m_iSize = 0;                   // zero the size of last
  ulUnknownBytes = 0L;           // zero the number of unknown bytes
  ulMyNumLargeUnknownBytes = 0L; // zero the number of large unknown bytes
  m_lTotalSize = 0L;             // zero the total size count
  OldNoutReturn = START;  // let last return = START

  SerDat.Clear ();        // Empty the serial data buffer.
}

void
NOutUnknown::ReSync (int iSize)
{
  m_lTotalSize += iSize;
  ulLastUnknownBytes += iSize;
  ulUnknownBytes += iSize;

  // We're lost, resync
  SerDat.ReSync (iSize);
```

```
}

INT
NOutUnknown::CopyStr(
   nBuffer& dat)
{
   for (m_iSize=1; m_iSize<MAX_NOUT_SIZE-1; m_iSize++)
   {
      // Get a character to identify the log
      if (dat.Get(&ucBuf[m_iSize], 1) == BUFFER_GET_EMPTY)
         return EMPTY;

      if (ucBuf[m_iSize] == '\n' || (ucBuf[m_iSize] < 31 || ucBuf[m_iSize]
> 126) && ucBuf[m_iSize] != 13)
         break;
   }

   // Did we find a line feed?
   if (ucBuf[m_iSize] != '\n' || m_iSize == MAX_NOUT_SIZE-1)
   {
      ReSync();         // We're lost, resync
      return(RESYNC);
   }

   ucBuf[++m_iSize] = '\0';  // Let's be nice to the user
   return(OKAY);
}

BUFFER_SET_RETURN
NOutUnknown::Set (unsigned char *ucbuf, INT iSize)
{
  BUFFER_SET_RETURN SetRet;      // The flag returned by the Set

  SetRet = SerDat.Set (ucbuf, iSize);

  return (SetRet);
}

NOUT_ID NOutUnknown::Id (void)   // Return the identity of the last read
log
{
  return iId;
}

const char *
NOutUnknown::GetIDStr (UINT uiLogID)
{
  //if we haven't already built it, build the m_ppszId array, based on the
map
  // <g_astLogMap>.
  if (NULL == m_ppszId)
    {
      // Get the total number of elements in the map.
      size_t uMapSize = sizeof (g_astLogMap) / sizeof (g_astLogMap[0]);

      // Allocate sufficient memory.
      m_ppszId = new char *[uMapSize];
      if (!m_ppszId)
       nxthrow (NCException, BadAlloc);
```

```
      // Build the array, based on the map.
      for (INT iEnumIndex = 0; iEnumIndex < MAX_NOUT_ID; iEnumIndex++)
       {
         m_ppszId[iEnumIndex] = NULL;

         // Find the corresponding element in the map
         for (UINT uMapIndex = 0;
              (uMapIndex < uMapSize) && (m_ppszId[iEnumIndex] == NULL);
              uMapIndex++)
           {
             if (g_astLogMap[uMapIndex].eID == iEnumIndex)
              m_ppszId[iEnumIndex] =
                const_cast < char *>(g_astLogMap[uMapIndex].szName);
           }

         // If we couldn't find it, ClassLib is screwed up.  Throw an
exception.
         if (NULL == m_ppszId[iEnumIndex])
           nxthrow_msg (NXOut, InvalidLogId,
                      "g_astLogMap is not mapping correctly");
       }
    }

  // Is it the default or the last record?
  if (ALL_NOUT_ID == uiLogID)
    uiLogID = iId;        // return that

  // Validate given iId.
  if (uiLogID >= MAX_NOUT_ID)
    nxthrow_msg (NXOut, InvalidLogId, "NTOut::szIdStr() invalid log id");

  return m_ppszId[uiLogID];      // return the log specified
}

//filename oem4d.cpp

#include <stdio.h>        /* Standard input/output definitions */
#include <string.h>       /* String function definitions */
#include <unistd.h>       /* UNIX standard function definitions */
#include <fcntl.h>        /* File control definitions */
#include <errno.h>        /* Error number definitions */
#include <sys/ioctl.h>
#include <sys/termios.h>
#include <sys/time.h>
#include <pthread.h>
#include "Buffer.hpp"
#include "GpsOutC.hpp"
#include "ntpshm.hpp"
#include "ConfigFile/ConfigFile.cpp"
#include "socket/Socket.h"
#include "socket/ServerSocket.h"
#include "socket/SocketException.h"
//#include "ConfigFile/Triplet.h"
//#define INITSTRING "\r\nunlogall\r\nlog rxstatusb once\r\nlog rawephemb
onchanged\r\nlog rangeb ontime 30\r\nlog bestposb ontime 600\r\nlog
rxconfiga once\r\nlog versionb once\r\n"
//#define INITSTRING "\r\nunlogall\r\nlog rawephema onchanged\r\nlog
almanaca onchanged\r\nlog rangea ontime 5\r\n"
```

```
//#define BAUDRATE B9600
//#define MODEMDEVICE "/dev/ttyS0"


pthread_cond_t* cond;
pthread_mutex_t* mutex;
struct termios oldtio, newtio;
void *read_function (void *null);
void *pop_function (void *null);
void *gpsd_ppsmonitor (void *null);
time_t mkgmtime(struct tm *t);
void Tokenize(const string& str,vector<string>& tokens);
void trim(string& str);
string CmdParse(string& cmd,int fn);
ConfigFile config( "Novatel.conf" );
string port;
int baudrate;
int fd;

NOut clParser;
NtpShm oNtp;
void read_conf(void){
       config.readInto( port, "port" );
       config.readInto( baudrate, "baudrate" );
       clParser.Obsinterval = config.read<int>( "Obsinterval" );
       clParser.ObsPrevInternval=clParser.Obsinterval;
       clParser.Navinterval = config.read<int>( "Navinterval" );
       clParser.NavPrevInternval=clParser.Navinterval;
       config.readInto( clParser.ObsFilesPath, "ObsFilesPath" );
       config.readInto( clParser.NavFilesPath, "NavFilesPath" );
}
int
open_port (void)
{
  int fn;
  /* File descriptor for the port */
  int status;
//      fd = open(MODEMDEVICE, O_RDWR );
  fn = open (port.c_str(), O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (fn == -1)
    {
      /*  *   * Could not open the port. * */
      puts ("open_port: Unable to open /dev/ttyS0 - ");
      exit(1);
    }
//  else
//Hardaware Reset
//////////////////////////
  ioctl (fn, TIOCMGET, &status);
  status &= ~TIOCM_RTS;
  ioctl (fn, TIOCMSET, &status);

//  fcntl(fn, F_SETFL, 0);
       fcntl(fn, F_SETFL, 0);
       cfsetispeed(&newtio, baudrate);
   cfsetospeed(&newtio, baudrate);
  tcgetattr(fn, &newtio);
  newtio.c_cflag &= ~PARENB;
  newtio.c_cflag &= ~CSTOPB;
```

```
  newtio.c_cflag &= ~CSIZE;
  newtio.c_cflag |= CS8;
  newtio.c_cflag |= (CLOCAL | CREAD);
  newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
  newtio.c_oflag &= ~OPOST;
  newtio.c_iflag = IGNPAR;
 // newtio.c_oflag = 0;
  //newtio.c_lflag = 0;       //ICANON;
//  newtio.c_iflag &= ~(IXON | IXOFF | IXANY);
//  newtio.c_cc[VMIN] = 1;
 // newtio.c_cc[VTIME] = 400;
 //
 // tcflush (fn, TCIFLUSH);
tcsetattr (fn, TCSANOW, &newtio);

/*
  tcgetattr (fn, &oldtio);
//              newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR;
//              newtio.c_oflag = 0;
//              newtio.c_lflag = ICANON;
  newtio.c_cc[VMIN] = 1;
  newtio.c_cc[VTIME] = 400;

//                   set the options
//  tcflush (fn, TCIFLUSH);
  */
  puts("Port Initialized...");
  return (fn);
}

int
close_port (int fn)
{
  tcsetattr (fn, TCSANOW, &oldtio);
  if (close (fn) == -1)
    puts ("Error");
}

int main ()
{
       cond = (pthread_cond_t*)malloc(sizeof(pthread_cond_t));
    mutex = (pthread_mutex_t*)malloc(sizeof(pthread_mutex_t));
       pthread_cond_init(cond, NULL);
    pthread_mutex_init(mutex, NULL);
       pthread_t thread1;
       pthread_t thread2;
       pthread_t thread3;
       int fa;
       ServerSocket server ( 30000 );
       //  pthread_mutex_init (&BUF.mutex, NULL);

       read_conf();
       fd = open_port ();
       //  write (fd, INITSTRING, strlen (INITSTRING));
       oNtp.ntpshm_init();
       if (oNtp.enable_ntpshm)
           oNtp.shmindex = oNtp.ntpshm_alloc();
       if (oNtp.shmindex >= 0 && oNtp.shmTimePPS){
```

```
                if ((oNtp.shmTimeP = oNtp.ntpshm_alloc()) >= 0)
                       pthread_create (&thread3, NULL, gpsd_ppsmonitor,
NULL);
        }
        if(pthread_create(&thread1, NULL, read_function, NULL) != 0){
                  puts("Could not create producer thread");

                  pthread_mutex_destroy(mutex);
                  pthread_cond_destroy(cond);

                  return(EXIT_FAILURE);
        }
        if(pthread_create(&thread2, NULL, pop_function, NULL) != 0){
                  puts("Could not create consumer thread");

                  pthread_mutex_lock(mutex);
                  //context.error = 1;
                  pthread_mutex_unlock(mutex);
                  pthread_cond_signal(cond);

                  pthread_join(thread1, NULL);

                  pthread_mutex_destroy(mutex);
                  pthread_cond_destroy(cond);;

                  return(EXIT_FAILURE);
        }
        //pthread_create (&thread1, NULL, read_function, NULL);
        //pthread_create (&thread2, NULL, pop_function, NULL);


        //write (fd, INITSTRING, strlen (INITSTRING));
        while ( true ){
               ServerSocket new_sock;
               cout<<"ss";
               server.accept ( new_sock );
               try{
                      while ( true ){
                             // pthread_mutex_lock( &mutex1 );
                             // cout<<"main locked";
                             std::string data;
                             // std::string command;
                             new_sock >> data;
                             // command="\r\n"+data+"\r\n";
                             new_sock <<CmdParse(data,fd);
                             //cout<<"-";
                             //cout<<data;
                             //new_sock << data;
                             // pthread_mutex_unlock( &mutex1 );
                             //cout<<"main unlocked";
                      }
               }
               catch ( SocketException& e) {
                      cout << "Exception was caught:" << e.description() <<
"\nExiting.\n";
               }

        }
```

```
       pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
       pthread_mutex_destroy(mutex);
       pthread_cond_destroy(cond);
       free(cond);
       free(mutex);
       close_port (fd);
       return 0;
}

void * send_message(void *null){

}

void *
read_function (void *null)
{
  //int fd;
  int nbytes = 0;
  int i=0;

  //fd = (int) ptr;
  unsigned char buff[255];
  string log;
  pthread_mutex_lock(mutex);
  while (1)
    {
            //cout<<"fun locked";
      nbytes = read (fd, buff, 255);
      clParser.Set (buff, nbytes);
      // Process all the logs that came in in that packet

      while (clParser.Read () == NOut::OKAY){
               if (clParser.Id () < LOGS_END)
                      printf ("Found Log: %s\r\n", clParser.szIdStr ());
               log.assign(clParser.szIdStr ());
               if (log=="TIMEB"){
                      oNtp.ntpshm_put(mktime(&clParser.ntp_time));
               }
        }
        pthread_cond_signal(cond);
      pthread_cond_wait(cond, mutex);

    }
       pthread_mutex_unlock(mutex);
}
void * pop_function (void *null){
       //pthread_mutex_lock( &mutex1 );
       pthread_mutex_lock(mutex);
        while (1){
              clParser.PopItems();
              pthread_cond_signal(cond);
              pthread_cond_wait(cond, mutex);
        }
        pthread_mutex_unlock(mutex);
       //pthread_mutex_unlock( &mutex1 );
}

void *gpsd_ppsmonitor(void *null)
```

```
{
    //struct gps_device_t *session = (struct gps_device_t *)arg;
    int cycle,duration, state = 0, laststate = -1, unchanged = 0;
    struct timeval tv;
    struct timeval pulse[2] = {{0,0},{0,0}};
    int pps_device = TIOCM_CTS;

    /* wait for status change on the device's carrier-detect line */
    while (ioctl(fd, TIOCMIWAIT, pps_device) == 0) {
       (void)gettimeofday(&tv,NULL);
       /*@ +ignoresigns */
       if (ioctl(fd, TIOCMGET, &state) != 0)
           break;
       /*@ -ignoresigns */

        state = (int)((state & pps_device) != 0);

       if (state == laststate) {
           if (++unchanged == 10) {
              cout<<"TIOCMIWAIT returns unchanged state, ppsmonitor
terminates\n";
              break;
           }
       } else {
            cout<<"pps-detect (CTS) on /dev/ttyS0 changed to "<<state<<"\n";
           laststate = state;
           unchanged = 0;
       }

       /*@ +boolint @*/


#define timediff(x, y)    (int)((x.tv_sec-y.tv_sec)*1000000+x.tv_usec-
y.tv_usec)
           cycle = timediff(tv, pulse[state]);
           duration = timediff(tv, pulse[state == 0]);
#undef timediff
       cout<<"PPS cycle: "<<cycle<<", duration: "<<duration<<"\n";
           if ( 800000 > duration) {
              /* less then 800mS, duration too short for anything */
                 cout<<"PPS pulse rejected too short. cycle: "<<cycle<<",
duration: "<<duration<<"\n";
           } else if (cycle > 999000 && cycle < 1001000 ) {
                 /* looks like PPS pulse */
              (void)oNtp.ntpshm_pps(&tv);
           } else if (cycle > 1999000 && cycle < 2001000) {
              /* looks like 2Hz square wave */
              (void)oNtp.ntpshm_pps(&tv);
            } else {
                 cout<<"PPS pulse rejected.  cycle: "<<cycle<<", duration:
"<<duration<<"\n";

           }

       /*@ -boolint @*/

       pulse[state] = tv;
    }
```

```
    return NULL;
}

time_t mkgmtime(struct tm *t)
/* struct tm to seconds since Unix epoch */
{
    int year;
    time_t result;
    const int cumdays[12] =
    {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};

    /*@ +matchanyintegral @*/
    year = 1900 + t->tm_year + t->tm_mon / 12;
    result = (year - 1970) * 365 + cumdays[t->tm_mon % 12];
    result += (year - 1968) / 4;
    result -= (year - 1900) / 100;
    result += (year - 1600) / 400;
    result += t->tm_mday - 1;
    result *= 24;
    result += t->tm_hour;
    result *= 60;
    result += t->tm_min;
    result *= 60;
    result += t->tm_sec;
    /*@ -matchanyintegral @*/
    return (result);
}

void trim(string& str)
{
  string::size_type pos = str.find_last_not_of(' ');
  if(pos != string::npos) {
    str.erase(pos + 1);
    pos = str.find_first_not_of(' ');
    if(pos != string::npos) str.erase(0, pos);
  }
  else str.erase(str.begin(), str.end());
}

void Tokenize(const string& str,vector<string>& tokens)
{
       string delimiters = " ";
    // Skip delimiters at beginning.
    string::size_type lastPos = str.find_first_not_of(delimiters, 0);
    // Find first "non-delimiter".
    string::size_type pos     = str.find_first_of(delimiters, lastPos);

    while (string::npos != pos || string::npos != lastPos)
    {
        // Found a token, add it to the vector.
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        // Skip delimiters.  Note the "not_of"
        lastPos = str.find_first_not_of(delimiters, pos);
        // Find next "non-delimiter"
        pos = str.find_first_of(delimiters, lastPos);
    }
}

string CmdParse(string &cmd,int fn){
```

```
	int i=0;
	vector<string> tokens;
	trim(cmd);
	Tokenize(cmd, tokens);
	string obsinterval;
	string almanac="\r\nlog almanacb onchanged\r\n";
	if(tokens.size()>0){
		if(tokens.at(0)=="exit")
			exit(1);
		else if (tokens.at(0)=="test")
		{
			cmd="\r\nlog unlogall\r\nlog almanacb onchanged\r\nlog
rangea ontime 30\r\nlog rangeb ontime 30\r\nlog rawephemb onchanged\r\nlog
rawephema onchanged\r\nlog gpsephema onchanged\r\n";
			write (fn, cmd.c_str(), strlen (cmd.c_str()));
			return "Wrote Command To Receiver : test";
		}
		else if (tokens.at(0)=="testnav")
		{
			cmd="\r\nlog unlogall\r\nlog rawephemb onchanged\r\nlog
rawephema onchanged\r\nlog gpsephema onchanged\r\n";
			write (fn, cmd.c_str(), strlen (cmd.c_str()));
			return "Wrote Command To Receiver : testnav";
		}
		else if(tokens.at(0)=="R"){
			cmd="\r\n"+cmd.substr (2,cmd.length())+"\r\n";
			if(tokens.size()>2){
				if(tokens.at(2)=="rangeb")
					write (fn, almanac.c_str(), strlen
(almanac.c_str()));
			}
			write (fn, cmd.c_str(), strlen (cmd.c_str()));
			return "Wrote Command To Receiver : " + cmd;
		}
		else if(tokens.at(0)=="P"){
			if(tokens.at(1)=="set"){
				if(tokens.at(2)=="obsinterval"){
					obsinterval=tokens.at(3);

	clParser.Obsinterval=atoi(obsinterval.c_str());
					return "Wrote Command To Receiver : " +
cmd;
				}
			}
		}

	}
	return "Please Recheck The Command!";
}


//filename oem4.cpp

#include "socket/Socket.h"
#include "socket/ClientSocket.h"
#include "socket/SocketException.h"
#include <iostream>
#include <string>
```

```
int main ( int argc, int argv[] )
{
  try{

      ClientSocket client_socket ( "localhost", 30000 );

      std::string reply;
         std::string command;
         char string[100];
       while(true){

      try{
                std::cout<<"OEM4 >>";
                //std::cin >> command;
                std::cin.getline(string,100,'\n');
                //command="log 1 2 3";
                //std::cout << string;
                client_socket << string;
                client_socket >> reply;
                //std::cout<<reply;
         }
      catch ( SocketException& ) {}

      std::cout << reply << "\n";

       }

  }
  catch ( SocketException& e )
    {
      std::cout << "Exception was caught:" << e.description() << "\n";
    }

  return 0;
}
```