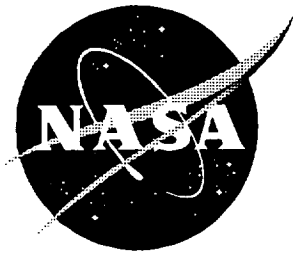
NASA Contractor Report 4752



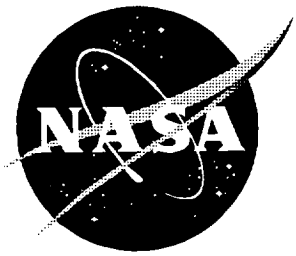
Using Formal Methods to Assist in the Requirements Analysis of the Space Shuttle GPS Change Request

Ben L. Di Vito and Larry W. Roberts

Contracts NAS1-19341 and NAS9-18817
Prepared for Langley Research Center

August 1996

NASA Contractor Report 4752



Using Formal Methods to Assist in the Requirements Analysis of the Space Shuttle GPS Change Request

Ben L. Di Vito
ViGYAN Inc.• Hampton, Virginia

Larry W. Roberts
Lockheed Martin Space Information Systems • Houston, Texas

National Aeronautics and Space Administration
Langley Research Center • Hampton, Virginia 23681-0001

Prepared for Langley Research Center
under Contracts NAS1-19341 and NAS9-18817

August 1996

Printed copies available from the following:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Contents

List of Figures

List of Tables

Executive Summary

We describe a recent NASA-sponsored pilot project intended to gauge the effectiveness of using formal methods in Space Shuttle software requirements analysis. Several Change Requests (CRs) have been selected as promising targets to demonstrate the utility of formal methods in this demanding application domain. A CR to add new navigation capabilities to the Shuttle, based on Global Positioning System (GPS) technology, is the focus of this industrial usage report. Portions of the GPS CR have been modeled using the language of SRI's Prototype Verification System (PVS). Developing the formal specifications, even with only limited analysis conducted on them, resulted in 86 requirements issues being discovered.

Experience with the GPS effort showed that the outlook for formal methods in this requirements analysis domain is quite promising. The approach detailed here produced specifications that requirements analysts (RAs) and others from the Shuttle community can and did learn to read and interpret, without having to become PVS practitioners. Moreover, the mere construction of formal specifications using this method can and did lead to the discovery of flaws in the requirements. There are good prospects for continuation of the effort by Shuttle personnel.

Future efforts can use the formal specifications constructed here as the foundation for more sophisticated analyses based on the use of formal proof. This additional tool provides the means to answer nontrivial questions about the specifications and achieve a higher level of assurance that the requirements are free of major flaws.

Based on the outcome of this case study, it appears that formalizing requirements would help overcome several deficiencies that have been cited for the current requirements analysis process:

1. *There is no methodology to guide the analysis.*

   Formal methods offer rigorous modeling and analysis techniques that bring increased precision and error detection to the realm of requirements.

2. *There are no completion criteria.*

   Writing formal specifications and conducting proofs are deliberate acts to which one can attach meaningful completion criteria.

3. *There is no structured way for RAs to document the results of their analysis.*

   Formal specifications are tangible products that can be maintained and consulted as analysis and development proceed. When provided as outputs of the analysis process, formalized requirements can be used as evidence of thoroughness and coverage, as definitive explanations of how CRs achieve their objectives, and as permanent artifacts useful for answering future questions and addressing future changes.

Chapter 1

Introduction

Among all the software developed by the National Aeronautics and Space Administration, Space Shuttle flight software is generally considered exemplary. Nevertheless, much of the requirements analysis and quality assurance activities in early lifecycle phases is done with products and tools of the late 1970s and early 1980s. Many activities remain manual exercises in need of more precise analysis techniques. Software upgrades to accommodate new missions and capabilities are continually introduced. Such upgrades underscore the need recognized in the NASA community, and in a recent assessment of Shuttle flight software development, for "state-of-the-art technology" and "leading-edge methodologies" to meet the demands of software development for increasingly large and complex systems [9, p. 91].

Over the last three years, NASA has investigated the use of formal methods (FM) in space applications, as part of a three-center demonstration project involving the Langley Research Center (LaRC), the Jet Propulsion Laboratory (JPL), and the Johnson Space Center (JSC). The goal of NASA's Formal Methods Demonstration Project for Space Applications is to find effective ways to use formal methods in requirements analysis and other phases of the development lifecycle [6, 7]. The Space Shuttle program has been cooperating in several pilot projects to apply formal methods to real-world requirements analysis activities such as upgrades supporting the recent MIR docking missions, improved algorithms for newly automated three-engine-out contingency abort maneuvers (3E/O) [2, 3], and the recent optimization of Reaction Control System Jet Selection (JS) [5].

We focus in this report on the formal methods-based analysis of a new Global Positioning System (GPS) navigation capability being added to the Shuttle [4]. This work was performed in the context of a broader program of formal methods activity at LaRC [1]. The effort consisted of formalizing selected Shuttle software (sub)system modifications and additions using the PVS specification language and interactive proof-checker [10]. Our objective was to explore and document the feasibility of formalizing critical Shuttle software requirements.

The key technical results of the project include a clear demonstration of the utility of formal methods as a complement to the conventional Shuttle requirements analysis process. The GPS project uncovered anomalies ranging from minor to substantive, many of which were undetected by existing requirements analysis processes.

1.1 Motivation

Since the late 1970s Shuttle software requirements have been and continue to be written using conventions known as Functional Subsystem Software Requirements (FSSRs) — low-level software requirements specifications written in English prose with strong implementation biases, and accompanied by pseudo-code, tables, and flowcharts. While the authors who create such requirements and the analysts who scrutinize them are very capable and diligent people, there are inherent limitations in the Shuttle program's requirements capture process. Due to the largely manual analysis methods, requirements surviving all the inspections and reviews still contain some residual flaws. Because errors that escape early detection are more costly to correct later, there is a definite gain to be realized by improving the quality of requirements analysis.

As will be discussed in Section 3.2.2, there are several well-known deficiencies in the existing requirements analysis process. Formal methods have been put forth as a possible way to address such deficiencies. It was the intention of this case study to help determine whether this prospect is realistic and to what extent. The GPS CR was chosen as a test case owing to its relatively large size and complexity. As a major augmentation of the Shuttle's avionics capability, the GPS CR contains enough complexity and poses enough development risk to make a feasibility assessment yield meaningful answers.

It was in this spirit that the GPS pilot project was undertaken. The results of the effort are detailed in the rest of this report.

1.2 Acknowledgments

The authors are grateful for the cooperation and support of many people during the course of this study. Several colleagues at Lockheed Martin Space Information Systems (formerly Loral Space Information Systems) were instrumental in getting the project off the ground and steering it in a useful direction. Among the Shuttle requirements analysts, Mike Beims and Bill McAllister had a hand in setting up the original pilot project and establishing the necessary framework for working with the GPS CR participants. J.H. Chuan provided preliminary overviews of the CR and its role within the Shuttle navigation system. In earlier phases of this work, two former Loral staff members played key roles: David Hamilton originally identified the GPS CR as a fruitful application for formal methods, and Dan Bowman coordinated the effort during its first phase.

Thanks are also due to other colleagues from around the country. John Rushby and Judy Crow from SRI International provided reviews of the formalization approach and guidance on the use of PVS. SRI's Sam Owre continued his vigilant maintenance of the PVS tools. Rick Butler of LaRC never wavered in his support of the effort and its potential benefit to the space program. John Kelly of JPL also was an enthusiastic supporter in his role as the overall coordinator of the Formal Methods Demonstration Project for Space Applications. Finally, we are indebted to Alice Robinson, formerly of NASA Headquarters, for her oversight and commitment during the three-year Demonstration Project.

This work was supported in part by the National Aeronautics and Space Administration under Contracts NAS1-19341 (ViGYAN) and NAS9-18817 (Lockheed Martin, formerly Loral).

Chapter 2

Formal Methods Background

Formal Methods (FM) consist of a set of techniques and tools based on mathematical modeling and formal logic that are used to specify and verify requirements and designs for computer systems and software. The use of FM on a project can assume various forms, ranging from informal specifications using English and occasional mathematical notation, to fully formal specifications using specification languages with a precise semantics. At their most rigorous, formal methods involve computer-assisted proofs of key system components or properties. Project managers choose from this spectrum of FM options as appropriate to optimize the costs and benefits of FM use and to achieve a level of verification that meets the project's needs and budget constraints. Formal methods play an important role in many activities including certification, reuse, and assurance.

Additional background and overview material on formal methods can be found in a recent NASA guidebook [8], excerpts of which appear below, and in Rushby's formal methods handbooks [11, 12].

2.1 Range of Formal Methods

Formal methods allow the logical properties of a computer system to be predicted from a mathematical model of the system by means of a logical calculation, which is a process analogous to numerical calculation. That is, formal methods make it possible to calculate whether a certain description of a system is internally consistent, whether certain properties are consequences of proposed requirements, or whether requirements have been interpreted correctly in the derivation of a design. These calculations provide ways of reducing or in some cases replacing the subjectivity of informal and quasi-formal review and inspection processes with a repeatable exercise. This is analogous to the role of mathematics in all other engineering disciplines; mathematics provides ways of modeling and predicting the behavior of systems through calculation. The calculations of FM are based on reasoning methods drawn mainly from formal logic. Systematic checking of these calculations may be automated.

Formal modeling of a system usually entails translating a description of the system from a non-mathematical model (data-flow diagrams, object diagrams, scenarios, English text, etc.) into a formal specification, using one of several formal languages. This results in a system description having a high degree of logical precision. FM tools can then be employed to logically evaluate this specification to reach conclusions about the completeness and consistency of the system's requirements or design. Manual analyses (e.g., peer reviews) of the formal model

are then used as an effective first check to assure the general reasonableness of the model. These are followed by tool-based analyses, which raise the level of reliability and confidence in the system specification even further. FM analysis techniques are based on deductive reasoning about system descriptions rather than statistical inferences drawn from system behavior, thereby allowing entire classes of issues to be resolved before requirements are committed to the design and implementation phases. Formal methods complement the systematic testing that follows implementation by allowing the testing phase to focus on a potentially smaller or more problematic range of test cases.

FM techniques and tools can be applied to the specification and verification of products from each development lifecycle: requirements, high-level and low-level design, and implementation. The process of applying FM to requirements or design differs mainly in the level of detail at which the techniques are applied. These techniques include: writing formal specifications, internal checking (e.g., parsing and type correctness), traceability checking, specification animation, and proof of assertions. Although this entire suite of techniques could be applied to all requirements and design elements, this is not the usual approach. Instead, an important subset of the requirements is chosen to undergo formal modeling, then a subset of the techniques is chosen for application. This enables the project to choose a level of verification rigor appropriate to its budget, schedule, and to the development team's technical needs.

In addition to the functions formal methods perform within a single development lifecycle phase, FM can also be used to establish and maintain strict traceability between system descriptions across different lifecycle phases. We can think of a hierarchy of system description documents, each of which describes the system at a different level of detail. Moving from the most abstract to the most concrete, there are requirements, high-level design, low-level design, and implementation. These documents also correspond to different lifecycle phases. FM can be used to demonstrate that a property at some level in the hierarchy gets implemented correctly by the next-lower level. In a thorough and rigorous treatment, FM can help demonstrate that requirements are correctly reflected in a subsequent design and that design features are correctly reflected in a subsequent implementation.

2.2 Formalization in the Early Lifecycle

While FM can be applied to any or all phases of the development lifecycle, the benefit-to-cost ratio for applying FM seems to be best during the requirements and high-level design phases. Formal methods complement early development phases, which are currently less automated, less tightly coupled to specific languages and notations, and their work products are typically less effectively analyzed than those of later stages of development. Formal methods compensate for these limitations without intruding on the existing process. For example, requirements are currently maintained as English language statements that are hard to check with automated tools. This deficiency is mitigated by the systematic, repeatable analysis supported by FM requirements specification and proof, while necessitating no changes to the natural language requirements statements.

Alternatively, as FM are injected deeper into the lifecycle, integration raises more technically challenging problems and the injection of FM becomes more intrusive. For example, the languages used for FM specification and proof and those used for programming generally exhibit fundamental semantic differences that make it difficult to synthesize a process that effectively uses both. Extreme care is required to ensure that such language differences are not themselves

a source of error during development.

A sound strategy is to apply FM to the earlier life-cycle phases where it will have the most positive impact. Incorporating FM will usually add a certain amount of cost to these phases while saving cost in later phases and during maintenance of the work products. In this respect, the use of FM is similar to other defect prevention techniques such as formal inspections. If heavy emphasis is already placed on analysis of early work products (e.g., requirements), the use of FM could potentially reduce lifecycle cost by augmenting or replacing ad-hoc techniques with more effective and systematic ones.

2.3 Prototype Verification System (PVS)

PVS is an environment for formal specification and verification developed at SRI International's Computer Science Laboratory [10]. The following sections are excerpted from SRI's World-Wide Web description of PVS located at URL `http://www.csl.sri.com/pvs/overview.html`. The system is freely available under license from SRI.

2.3.1 The PVS System

PVS consists of a specification language, a number of predefined theories, a theorem prover, various utilities, documentation, and several examples that illustrate different methods of using the system in several application areas. PVS exploits the synergy between a highly expressive specification language and powerful automated deduction; for example, some elements of the specification language are made possible because the typechecker can use theorem proving. This distinguishing feature of PVS has allowed perspicuous and efficient treatment of many examples that are considered difficult for other verification systems.

2.3.2 The PVS Language

The specification language of PVS is based on classical, typed higher-order logic. The base types include uninterpreted types that may be introduced by the user, and built-in types such as the booleans, integers, reals, and the ordinals up to ϵ_0; the type-constructors include functions, sets, tuples, records, enumerations, and recursively-defined abstract data types. Predicate subtypes and dependent types can be used to introduce constraints, such as the type of prime numbers. These constrained types may incur proof obligations (conditions to be established by theorem proving) during typechecking, but greatly increase the expressiveness and naturalness of specifications. In practice, most of the obligations are discharged automatically by the theorem prover. PVS specifications are organized into parameterized theories that may contain assumptions, definitions, axioms, and theorems. Definitions are guaranteed to provide conservative extension; to ensure this, recursive function definitions generate proof obligations. PVS expressions provide the usual arithmetic and logical operators, function application, lambda abstraction, and quantifiers, within a natural syntax. Names may be freely overloaded, including those of the built-in operators such as AND and +.

2.3.3 The PVS Prover

The PVS theorem prover provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The prim-

itive inferences include propositional and quantifier rules, induction, rewriting, and decision procedures for linear arithmetic. The implementations of these primitive inferences are optimized for large proofs: for example, propositional simplification uses binary decision diagrams (BDDs), and auto-rewrites are cached for efficiency. User-defined procedures can combine these primitive inferences to yield higher-level proof strategies. Proofs yield scripts that can be edited, attached to additional formulas, and rerun. This allows many similar theorems to be proved efficiently, permits proofs to be adjusted economically to follow changes in requirements or design, and encourages the development of readable proofs.

2.3.4 The PVS Interface

PVS uses Gnu Emacs to provide an integrated interface to its specification language and prover, and provides many status-reporting and browsing tools, as well as the ability to generate typeset specifications (in user-defined notation) using LaTeX.

2.3.5 PVS Applications

PVS is mainly intended for the formalization of requirements and design-level specifications, and for the analysis of intricate and difficult problems. It (and its predecessors) have been chiefly applied to algorithms and architectures for fault-tolerant flight control systems, and to problems in hardware and real-time system design. Several examples are described in papers listed on the PVS home page. Collaborative projects involving PVS are ongoing with NASA and several aerospace companies; applications include a microprocessor for aircraft flight-control, diagnosis and scheduling algorithms for fault-tolerant architectures, and requirements specification for portions of the Space Shuttle flight-control system.

PVS has been installed at over 30 sites in North America, Europe, and Asia; current work is developing PVS methodologies for highly automated hardware verification (including integration with model checkers), and for concurrent and real-time systems (including a transparent embedding of the duration calculus, a specialized formalism for modeling time).

Chapter 3

Shuttle Software Background

NASA's prime contractor for the Space Shuttle is the Space Systems Division of Rockwell International. Lockheed Martin Space Information Systems (formerly Loral Space Information Systems) is their software subcontractor for both onboard Shuttle software as well as support functions. Draper Laboratory also serves Rockwell, providing requirements expertise in guidance, navigation and control. Flight software requirements are written by Rockwell with significant support from Draper. New requirements are submitted to Lockheed Martin for analysis before proceeding to the development phase. After Shuttle community review and approval, the final requirements are accepted and become binding.

3.1 Software Organization

Shuttle flight software executes in four redundant general purpose computers (GPCs), with a fifth backup computer carrying dissimilar software. Much of the Shuttle software is organized into major units called *principal functions*, each of which may be subdivided into *subfunctions*. Since the late 1970s software requirements have been and continue to be written using conventions known as Functional Subsystem Software Requirements (FSSRs) — low-level software requirements specifications written in English prose with strong implementation biases, and accompanied by pseudo-code, tables, and flowcharts. Interfaces between software units are specified in input-output tables. Inputs can be variables or one of three types of constant data: *I-loads* (fixed for the current mission), *K-loads* (fixed for a series of missions), and physical constants (never changed).

Shuttle software modifications are packaged as Change Requests (CRs), that are typically, but not always, modest in scope, localized in function, and intended to satisfy specific needs for upcoming missions. Roughly once a year, software releases called Operational Increments (OIs) are issued incorporating one or more CRs. Shuttle CRs are written as modifications, replacements, or additions to existing FSSRs. Lockheed Martin Requirements Analysts (RAs) conduct thorough reviews of new CRs, analyzing them with respect to correctness, implementability, and testability before turning them over to the development team. Their objective is to identify and correct problems in the requirements analysis phase, avoiding far more costly fixes later in the lifecycle.

The formalization approach is based on several assumptions about the requirements expressed in the GPS CR and about the Shuttle software architecture in general. These assumptions are enumerated below.

3.1.1 Execution Environment

The following assumptions are made about those aspects of the Shuttle execution environment pertinent to the GPS CR:

1. A *principal function* is a software subsystem executed periodically at a mode-dependent rate such as 6.25 Hz.
2. A principal function is a virtual software entity that is decomposed into several *subfunctions.*
3. The subfunctions within a principal function are executed sequentially in the order they are presented in the requirements.
4. Outputs produced by an earlier subfunction may be passed as inputs to a subfunction executed later within the same principal function invocation.

3.1.2 Principal Function Inputs and Outputs

Several key attributes about principal function inputs and outputs are noted below:

1. The external interface for a principal function is specified by tables of inputs and outputs.
2. Data types are identified in the input-output tables.
3. No external data objects other than those listed in the tables may be accessed.
4. Some input and output values represent "state" information, that is, data saved for consumption by the next execution of the principal function, such as delay elements (z^{-1}).
5. Any "backward" data flows from a later subfunction to an earlier one do not take place within the same principal function invocation and are saved as state information for the next iteration.

3.1.3 Subfunction Inputs and Outputs

Subfunctions also have input and output characteristics worth noting:

1. Subfunction inputs may be either 1) "passed down" from principal function inputs, 2) retrieved as components of the previous state, or 3) passed from earlier subfunction outputs.
2. Subfunction outputs may be either 1) "passed up" to principal function outputs, 2) saved as components of the next state, or 3) passed to later subfunction inputs.
3. No external data objects other than those listed in the tables may be accessed.
4. Data types are identified in the input-output tables.

3.1.4 Local Variables

Subfunction requirements may refer to local variables to describe the required processing steps.

1. Local variables within requirements are not explicitly identified as such; this status is implied by their absence from the input-output tables.
2. Some local variables are accessible only within a single subfunction execution; there is no persistence for such variables.
3. Other local variables serve as state variables that persist until the next time the subfunction is invoked.
4. Data types are not provided for locals; types must be inferred from context.

3.2 Requirements Analysis

The Space Shuttle flight software (FSW) project has a well-defined process for performing Requirements Evaluation (RE). This process is responsible for ensuring that requirements generated by an engineer are complete, consistent, implementable, and will solve the problem that the engineer wants to solve. However, this process does not include a well-defined set of analysis methods or techniques. That is, while there is a well-defined process, there is not a well-defined set of methods for accomplishing the evaluation.

3.2.1 The Change Request (CR) Process

When an engineer recognizes a need to change or add a capability to a Shuttle flight software system, he or she will author a requirements Change Request (CR). Typically, the author sends early drafts of a CR to a FSW requirements analyst (RA) who performs an informal review of the CR and returns comments to the author. Once the author feels confident that the requirements are correct, he or she submits them to a review board. After all the submitted CRs are prioritized, those of highest priority are formally reviewed by the FSW RAs in a process that includes the following:

- The preparation of an engineering assessment that contains a summary of the change, why it is needed, and its potential impact on the software system.
- An in-depth analysis of the CR guided by a Requirements Inspection Checklist that contains descriptions of generic error categories. RAs look for instances of these error types in the CR. If they find anything that they consider erroneous, they fill out Issue Forms to document the potential errors.
- A formal inspection of the CR (large and complex CRs may require multiple inspections) that consists of one or more meetings of responsible FSW and Shuttle community members to review all the issues that have been found during analysis of the CR and to compile a list of issues that must be investigated before the CR can be declared ready for implementation. An additional purpose of the inspection is to ensure that all participants (author, RA, developer, verifier, et al.) have a consistent understanding of the requirements.

- Tracking and resolution of all issues. Each Issue Form remains "open" until it is determined that the issue is not a problem, or that it is a problem and a solution has been found; then it is "closed."

- Baselining. When all required inspections have been held for a CR and all issues have been closed, a CR is declared ready for implementation by a control board. Then it is baselined and scheduled for implementation.

3.2.2 Deficiencies in the Current Process

The step in the existing process that is of most interest to this case study is the in-depth analysis of a CR. This step involves studying, understanding, and analyzing the CR. There are currently three major deficiencies in the analysis step.

1. *There is no methodology to guide the analysis.*

   As described by one RA, this step is "unstructured and very dependent on the background, intelligence, and perseverance of the RA." To address this deficiency, the Shuttle project is currently trying to identify methods that skilled RAs can use.

2. *There are no completion criteria.*

   The project recognizes this as a key deficiency, but currently does not have a plan for addressing it.

3. *There is no structured way for RAs to document the results of their analysis.*

   This causes several problems:

   - The only evidence that RAs have to show thoroughness in their analysis is the number of issues found. RAs feel that this is not a good measure of what they do.
   - When a CR is analyzed, the RA gains an understanding of how the CR works and why it is or is not correct. Without a structured way to document the analysis that was performed, this understanding is lost. The Shuttle project is trying to address this drawback by encouraging RAs to make public whatever notes they jot down while doing their analysis.
   - There is no evidence that demonstrates the benefit of quality assurance on the CR. The Issue Forms only show what is wrong with a CR. There is nothing to show what is right about the CR.

3.2.3 Quality Metrics to Assess the Current CR Process

The existing CR process also includes the collection and analysis of quality metrics, primarily the number of issues collected during inspection and the number of problems found in a CR after requirements evaluation has been completed (these problems are said to have "escaped" the RE process). The ratio of the number of escaped problems to the total number of issues found is a measure of the quality (or effectiveness) of the RE process; this ratio is called the process error rate. The process error rate has been gradually improving (i.e., getting smaller) in recent years, but is still far worse than the process error rate for design and code (i.e., the ratio of problems that escape design/code review to the total number of issues found during

design/code). This situation indicates that the methods used for requirements evaluation are not as effective as those used for design and code (this should not be surprising given the absence of defined methods for requirements analysis). Not only is the RE process less effective, but the data captured on software problems show that there are more errors injected into requirements than are injected into design/code (i.e., there are more requirements errors to be found than design or code errors to be found). As a result, most of the problems encountered using the Shuttle software are caused by requirements errors, not software design or code errors.

Chapter 4

GPS Change Request

As one of the larger ongoing Shuttle Change Requests (CRs), the Global Positioning System (GPS) CR involves a significant upgrade to the Shuttle's navigation capability. Shuttles are to be outfitted with GPS receivers and the primary avionics software will be enhanced to accept GPS-provided positions and integrate them into navigation calculations. Prior to implementing the CR, requirements analysts at Lockheed Martin Space Information Systems, the Shuttle software contractor, must scrutinize the CR to identify and resolve any requirements issues.

4.1 Global Positioning System (GPS) Navigation

GPS is a satellite-based navigation system operated by the U.S. Department of Defense (DoD). It consists of a constellation of 24 satellites in high earth orbits. Navigation is effected using a receive-only technique. Dedicated hardware receivers track four or more satellites simultaneously and recover their signals from the code division multiplexing inherent in their method of transmission. Receivers solve for position and velocity, with a horizontal position accuracy of 100 meters for the Standard Positioning Service mode of operation.

The GPS retrofit to the Shuttle was planned in anticipation of TACAN, a ground-based navigation system currently used during entry and landing, being phased out by the DoD. Originally, GPS was required for navigation only during the entry flight phase after the disappearance of TACAN, but the scope has been broadened to cover all mission phases. This makes the GPS CR a rather significant upgrade to the Shuttle's navigation capability. Shuttle avionics hardware will be augmented with a set of suitable GPS receivers, and the primary avionics software will be enhanced to accept and process GPS-provided positions, making them available to various onboard navigation functions. In particular, the GPS CR will provide the capability to update the Shuttle navigation filter states with selected GPS state vector estimates similar to the way state vector updates currently are received from the ground. In addition, the new functions will provide feedback to the GPS receivers and will support crew control and operation of GPS/General Purpose Computer (GPC) processing.

4.2 Characteristics of Application

The nature of the GPS CR application is that of a significant augmentation to a mature body of complex navigation functions. Interfaces among components are broad, containing many variables. Typical classes of data include:

- Flags to indicate status, to request services, and to select options among processing choices.
- Time values and time intervals to serve as timestamps within state vectors and as integration intervals, and to control when operations should be performed.
- Navigation-related positions and velocities, usually in the form of three-element vectors.
- Arrays of all these types indexed by GPS receiver number.
- Various numeric quantities representing thresholds, tolerance values, etc.

Navigation state vectors are of the form $(\mathbf{r}, \mathbf{v}, t)$, where $\mathbf{r}$ is a position, $\mathbf{v}$ is a velocity, and t is the time at which the position and velocity apply. A position $\mathbf{r}$ or a velocity $\mathbf{v}$ is a three-element vector relative to a Cartesian coordinate system. Usually the Shuttle uses an inertial coordinate system called the "Aries mean of 1950" system, abbreviated as "M50."

An important operation on state vectors is *propagating* them to a new instant of time. If we have a state vector $(\mathbf{r}, \mathbf{v}, t)$, and we have a measurement or estimate of the accelerations experienced by the vehicle over the (short) time interval $[t, t']$, we can propagate the state to a new state vector $(\mathbf{r}', \mathbf{v}', t')$ using standard techniques of physical mechanics. This type of operation is typically performed to synchronize state vectors to a common point in time.

Processing requirements within the CR are generally expressed in an algorithmic style using high-level language assignments and conditional statements. Within conditionally invoked assignments, the assumption is the usual procedural one that a variable not assigned retains its previous value, which may or may not have a meaningful interpretation in the current context. Flag variables are used to indicate when other variables hold currently valid data.

4.3 Enhanced Shuttle Navigation System

The GPS upgrade is being conducted according to a two-phase integration plan. First, a single-string implementation will be carried out involving only a single GPS receiver. After a test and evaluation period, the full-up implementation involving three receivers will provide the operational configuration. Software changes are designed to support a single receiver while accommodating the three-receiver setup from the outset, thus requiring fewer additional changes to migrate to the full-up version.

Figure 4.1 shows the integrated architecture for the enhanced navigation subsystem. GPS receivers are managed by the GPS Subsystem Operating Program (SOP), which acts as a device driver. The new principal function GPS Receiver State Processing accepts GPS state vectors, and selects and conditions a usable one for presentation to the appropriate navigation user. Another new principal function, GPS Reference State Processing, maintains reference states for the receivers. Inertial measurement units (IMUs) provide acceleration data, and Redundancy Management (RM) functions maintain failure status information.

4.4 Subset Chosen for Formal Specification

The GPS CR under scrutiny addresses only the single GPS receiver implementation. Nevertheless, our GPS formalization focused on a subset of the functionality because this CR is still large and complex. After preliminary study of the CR and discussions with the GPS requirements

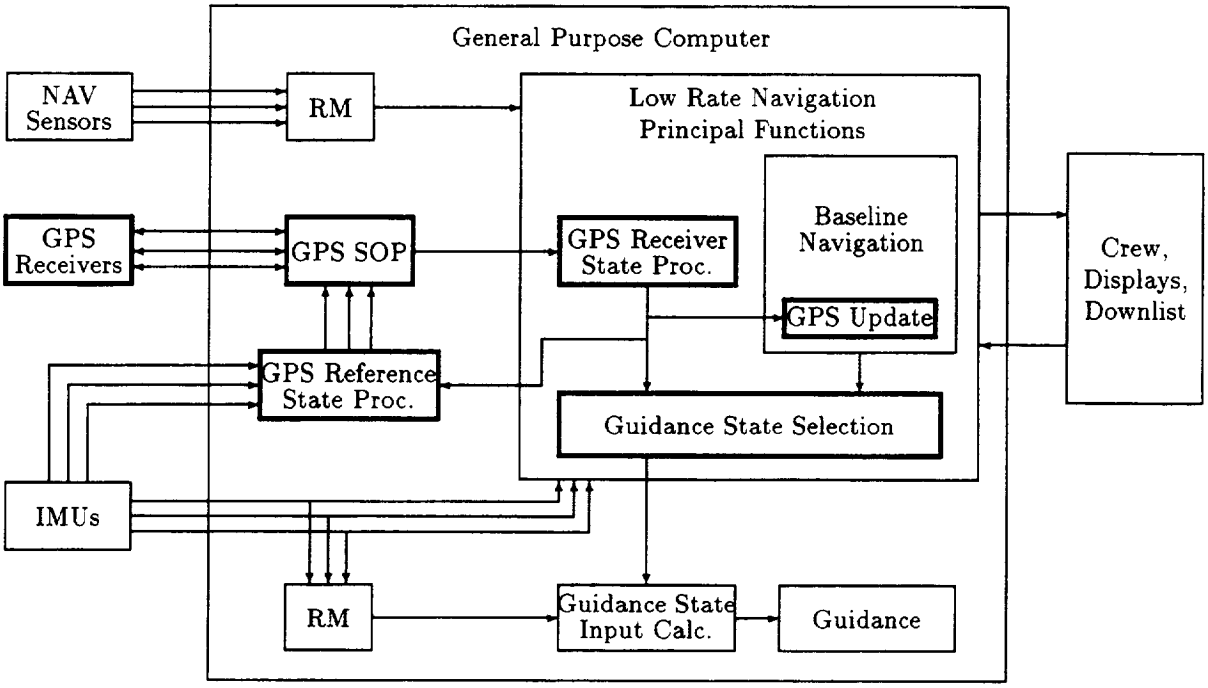


Figure 4.1: Architecture for integrating GPS into navigation subsystem.

analysts, we decided to concentrate on two major new principal functions, emphasizing their interfaces to existing navigation software and excluding crew display interface functions. The two principal functions, known as GPS Receiver State Processing and GPS Reference State Processing, select and modify GPS state vectors for consumption by the existing entry navigation software.

These two principal functions, in turn, are organized into several subfunctions each. The chosen subset of the GPS CR encompasses approximately 110 pages of requirements in the form of prose, pseudo-code, tables, and flowcharts. The entire CR is somewhat over 1000 pages.

4.4.1 GPS Receiver State Processing

The subfunctions of GPS Receiver State Processing are as follows:

1. GPS IMU Assign
2. GPS Navigation State Propagation
3. GPS State Vector Quality Assessment
4. GPS State Vector Selection
5. GPS Reference State Announced Reset

6. GPS Downlist Computation

Subfunctions (2), (3), and (4) involve fairly complex processing to evaluate, select, and modify usable GPS state vectors. The other subfunctions have more modest functionality.

4.4.2 GPS Reference State Processing

The subfunctions of GPS Reference State Processing are as follows:

1. GPS External Data Snap
2. IMU GPS Selection
3. GPS Reference State Initialization and Reset
4. GPS Reference State Propagation

Of these, only subfunctions (3) and (4) contain any significant complexity.

Chapter 5

Technical Approach

The formal methods approach is loosely based on earlier work conducted by the inter-center team during 1993 on the Shuttle subsystem Orbit Digital Auto-pilot (DAP). Those techniques were adapted to accommodate the needs of GPS for the Shuttle software. All work has been mechanically assisted by the PVS toolset [10]. The distinguishing characteristic of PVS is a highly expressive specification language coupled with a very effective interactive theorem prover that uses decision procedures to automate most of the low-level proof steps.

5.1 State Machine Models

We have devised a strategy to model Shuttle principal functions based on the use of a conventional abstract state machine model. Each principal function is modeled as a state machine that takes inputs and local state values, and produces outputs and new state values. This method provides a simple computational model similar to popular state-based methods such the A-7 model [13].

One transition of the state machine model corresponds to one scheduled execution of the principal function, e.g., one cycle at rate 6.25 Hz or other applicable rate. All of the inputs to the principal function are bundled together and a similar bundling of the outputs is arranged. The state variable holds values that are (usually) not delivered to other units, but instead are held for use on the next cycle.

The state machine transition function is a mathematically well-defined function that takes a vector of input values and a vector of previous-state values, and maps them into a vector of outputs and a vector of next-state values.

$$M : I \times S \rightarrow [O \times S]$$

This function M is expressed in PVS and forms the central part of the formal specification. We construct a tuple composed of the output and state values so only a single top-level function is needed in the formalization. Some values may appear in both the output list and the next-state vector.

While the function M captures the functionality of the software subsystem in question, the state machine framework can also serve to formalize abstract properties about the behavior of the subsystem. The common approach of writing assertions about *traces* or sequences of input and output vectors is easily accommodated. For example, we can introduce sequences $I(n) = < i_1, \ldots, i_n >$ and $O(n) = < o_1, \ldots, o_n >$ to denote the flow of inputs and outputs

that would have occurred if the state machine were run for n transitions. A property about the behavior of M can be expressed as a relation P between $I(n)$ and $O(n)$ and formally established, i.e., we prove that the property P does indeed follow from the formal specification M using the PVS proof-checker.

5.2 PVS Techniques

The basic structure of the formal specifications can be created using only a few of the PVS language features. PVS specifications are organized around the concept of *theories*. Each theory is composed of a sequence of declarations or definitions of various kinds. Definitions from other theories are not visible unless explicitly imported.

Structured data types or record types are used extensively in specifications. These types are introduced via declarations of the following form:

```
record_type: TYPE = [# v1: type_1, v2: type_2, ... #]
```

A component of a record may be accessed using the notation `v1(R)`. A record value constructed from individual component values may be synthesized as follows:

```
(# v1 := <expression 1>, v2 := <expression 2>, ... #)
```

Logical variables are introduced to serve as arguments to functions and to express logical formulas or assertions:

```
x, y, z: VAR var_type
```

Local variable declarations also are available in most cases, but it is often clearer to enumerate variables and their types more globally. Variable declarations apply throughout the containing theory but no further.

A function is defined quite simply by the following notation:

```
fn (arg_1, arg_2, ...): result_type =  <expression>
```

Each of the variables `arg_i` must have been declared of some type previously. The definition must be mathematically well-defined, meaning its single result value is a function of the arguments and possibly some constants. No "free" variables are allowed within the expression. In addition, the type of the expression must be compatible with the result type.

Besides fully defining functions, it is possible to declare unspecified functions using the notation:

```
fn (arg_1, arg_2, ...): result_type
```

In this case, the function's signature is provided, but there is no definition. This is often useful when developing specifications in a top-down fashion. Also, it may be that some functions will never become defined in the specification, in which case they can never be expanded during a proof.

One type of expression in PVS is particularly useful for the kind of specifications envisioned here. This feature, known as a LET expression, allows the introduction of bound variable names to refer to subexpressions.

```
pf_result: TYPE = [# output: pf_outputs, state: pf_state #]

principal_function (pf_inputs, pf_state,
                    pf_I_loads, pf_K_loads,
                    pf_constants) : pf_result =

         (# output := <output expression>,
            state  := <next-state expression>
          #)
```

Figure 5.1: PVS model of a Shuttle *principal function.*

```
LET v1 = <expression 1>, v2 = <expression 2>, ...
IN  <expression involving v1, v2, ...>
```

Each of the variables serves as a shorthand notation used in the final expression. The meaning is the same as if each of the subexpressions were substituted for its corresponding variable.

5.3 Specification Approach

The state machine formalization can be realized using PVS in a variety of ways. One method is outlined here based on the PVS facility for defining mathematically rigorous functions. It is a scheme that prescribes an orderly structure for the notation, and allows a systematic progression from one layer of specification to the next to manage the complexity of detailed requirements specifications.

5.3.1 Specifying Principal Functions

At the highest level, we would like the specification of a principal function to have the general form shown in Figure 5.1. This function is an abstract PVS rendering of the state machine model described in section 5.1. The definition assumes all input and state values have been collected into the structures `pf_inputs` and `pf_state`. Additionally, all I-load, K-load, and constant inputs used by the principal function are collected into similar structures. The `pf_result` type is a record that contains an output component and a next-state component. Each of these objects is, in turn, a structure containing (possibly many) subcomponents.

The output and next-state expressions in the general form above must describe the effects of invoking the subfunctions belonging to the principal function. In practice, this can be very complicated so a stylized method of organizing this information has been devised. It is based on the use of a LET expression to introduce variable names corresponding to the intermediate inputs and outputs exchanged among subfunctions.

We begin with several type declarations for the structured data types needed by the scheme. These types, shown in Figure 5.2, implement the bundling of inputs, outputs, and state variables discussed earlier. They help define the interface of the principal function being specified. Similar types are used to model the collective outputs of subfunctions.

The principal function itself can be specified by characterizing its outputs and next-state values in a systematic manner. A general scheme for doing this is sketched in Figure 5.3. In this

```
pf_inputs: TYPE = [#
      input_1:       input_1_type,
      input_2:       input_2_type,
      . . .
      #]

pf_state: TYPE = [#
      state_var_1:   state_1_type,
      state_var_2:   state_2_type,
      . . .
      #]

pf_I_loads: TYPE = [#
      I_load_1:      I_load_1_type,
      I_load_2:      I_load_2_type,
      . . .
      #]

pf_K_loads: TYPE = [#
      K_load_1:      K_load_1_type,
      K_load_2:      K_load_2_type,
      . . .
      #]

pf_constants: TYPE = [#
      constant_1:    constant_1_type,
      constant_2:    constant_2_type,
      . . .
      #]

pf_outputs: TYPE = [#
      output_1:      output_1_type,
      output_2:      output_2_type,
      . . .
      #]

pf_result: TYPE = [# output: pf_outputs, state: pf_state #]
```

Figure 5.2: State machine interface types.

```
principal_function (pf_inputs, pf_state,
                    pf_I_loads, pf_K_loads,
                    pf_constants) : pf_result =

  LET
      subfun_1_out =
        subfunction_1(input_1(pf_inputs),
                      input_2(pf_inputs), ... ,
                      state_var_1(pf_state),
                      state_var_2(pf_state), ... ,
                      I_load_1(pf_I_loads),
                      I_load_2(pf_I_loads), ... ,
                      K_load_1(pf_K_loads),
                      K_load_2(pf_K_loads), ... ,
                      constant_1(pf_constants),
                      constant_2(pf_constants), ... ),

      subfun_2_out =
        subfunction_2(input_1(pf_inputs),
                      input_2(pf_inputs), ... ,
                      state_var_1(pf_state),
                      state_var_2(pf_state), ... ,
                      I_load_1(pf_I_loads),
                      I_load_2(pf_I_loads), ... ,
                      K_load_1(pf_K_loads),
                      K_load_2(pf_K_loads), ... ,
                      constant_1(pf_constants),
                      constant_2(pf_constants), ... ,
                      sf_1_output_1(subfun_1_out),
                      sf_1_output_2(subfun_1_out), ... ),

        . . .

  IN
      (# output := (# output_1 := sf_i_output_j(subfun_i_out),
                      output_2 := sf_k_output_l(subfun_k_out),
                       . . . #),

         state := (# state_var_1 := sf_m_output_n(subfun_m_out),
                     state_var_2 := sf_p_output_q(subfun_p_out),
                       . . . #)
      #)
```

Figure 5.3: General form of principal function specification.

scheme, a LET expression is used to give names to the intermediate data objects produced by the subfunction invocations. These output variable names can then be accessed in the defining expressions for the overall principal function output and next-state structures.

Four categories of data are representable using the principal function arguments and the various LET variables:

1. External inputs to the principal function extracted as individual components of the collection `pf_inputs`.

2. Previous-state inputs to the principal function extracted as components of the collection `pf_state`.

3. Data constants used by the principal function extracted as components of the collections `pf_I_loads`, `pf_K_loads`, and `pf_constants`.

4. Values representing the individual subfunction outputs extracted as components of the appropriate collection, `subfun_i_out`.

Note that a variable introduced using a LET expression may be referenced at any later point in the LET expression. Thus, a variable may be given a value derived from the values of previously listed variables within the same LET expression. This feature is used to provide the outputs of one subfunction as inputs to later subfunctions. In Figure 5.3, for example, the subfunction outputs are available in LET variables `subfun_1_out`, `subfun_2_out`, etc. The jth output of subfunction i is represented schematically by the expression `sf_i_output_j(subfun_i_out)`.

5.3.2 Specifying Subfunctions

A subfunction specification has the form of a straightforward PVS function definition plus a data type to achieve the grouping of subfunction outputs. Figure 5.4 shows the schematic form. The arguments to the function are inputs from one of four possible sources:

1. Principal function inputs passed down to the subfunction.

2. State variables passed down to the subfunction.

3. Data constants passed down to the subfunction.

4. Intermediate results appearing as outputs of other subfunctions.

A subfunction may use any combination of inputs from these sources. These categories are introduced strictly for the purpose of modeling the underlying software structure. There is nothing that distinguishes them through their representation in PVS.

Several other aspects of this specification style are noteworthy:

- The subfunction outputs are grouped together using a result type based on a record structure, but the subfunction inputs are treated individually. An alternative would be to group both inputs and outputs. Either arrangement is possible. It seems like the scheme with separate inputs is more direct and more effective at describing the structures of interconnected subfunctions that are typically encountered.

```
subfun_outputs: TYPE = [#
      output_1:       output_1_type,
      output_2:       output_2_type,
      . . .
      #]

subfunction (input_1, input_2, ...): subfun_outputs =

    (# output_1 := < output expression 1 >,
       output_2 := < output expression 2 >,
        . . .
    #)
```

Figure 5.4: General form of subfunction specification.

- No special provision is made for state variables at the subfunction level. Next-state values are included in the list of outputs along with ordinary output values. Their treatment as state values is handled at the next higher level in the principal function specification. Again, the alternative scheme for handling this, namely, separating out the next-state values, is also possible. The method chosen reduces some of the data structure manipulation that otherwise would be necessary.

- With this scheme and that for the principal function specifications, it is usually unnecessary to distinguish previous-state from next-state values using a variable naming convention. The appropriate role is determined by the context. However, if an input is named something of the form `prev_value` in the requirements themselves, we will naturally use this name to maintain the correspondence.

If the subfunction is to be unspecified, a stub for it may be created by omitting the PVS function's defining expression:

```
subfunction (input_1, input_2, ...): subfun_outputs
```

This form would allow the principal function to be specified in PVS without requiring the full details of the subfunctions to be provided. Such unspecified functions can be elaborated later if desired.

Output expressions for the subfunction specifications are simply functional expressions involving the input variables such as:

```
GPS_off_count + 1
```

Often the operations will be more complicated and rely on the use of auxiliary functions to formalize the concepts:

```
dot_product(vector_1, scalar_mult(alpha, vector_2))
```

While the PVS specification language does not have operators like dot product as built-in language primitives, it is a simple matter to introduce them as user-defined functions. In this way, it is possible to formalize the relevant concepts and build up whatever operators are suitable for the task at hand.

5.4 Theory Organization

Large specifications written in PVS are usually organized into a number of individual theories. Each theory is an entity with scoping rules similar to modules or packages in modern programming languages. Types, constants, functions, and a few other PVS constructs may be imported from other theories through the use of explicit importation directives. This is usually done by importing other theories in their entirety, thereby making all of their symbols visible locally. Variables, however, are not exportable; they can be used only within a single theory.

The PVS toolset provides various commands for operating on both theories and files. A PVS specification file may contain more than one theory. Nevertheless, it is generally advisable to maintain a strict usage convention of having only one theory per file. This approach avoids unwelcome complications that can arise from working with a collection of multiple-theory files. No loss of capability is incurred by adhering to this convention.

Several considerations are worth keeping in mind when developing PVS specifications and organizing them into theories.

1. As in programming, it is natural to develop common theories of types, constants, and utility functions that may be imported by many other theories. It may be necessary to structure these as hierarchies to maintain a structure parallel to that of the specification theories they serve.

2. One theory should be dedicated to the top-level specification for the principal function.

3. Subfunction specifications should be allocated to separate theories. It may be possible to collect them all into a single theory, especially if they are only stubbed out and not fully defined, but in the long run it will be worthwhile to separate them.

4. Subordinate functions used to express specifications and imported by several of these separate theories may occupy a middle position in the theory hierarchy.

5. If the formal specification effort reaches the point of carrying out mechanical proofs, additional theories need to be included. Such theories are needed to enumerate the various axioms, lemmas, and theorems required as well as auxiliary support for the proofs themselves.

5.5 Deviations from CR/FSSR Requirements

In deriving the preceding specification method, we have tried to be faithful to the FSSR method of expressing requirements. A few deviations and omissions, however, should be noted.

- The concept of state variables is not explicitly mentioned in FSSR-style requirements. Their use has been inferred and a method has been provided for their specification to make the final requirements more clear.

- No provision was introduced to capture initialization requirements for state variables. This issue can be handled at the next higher level of modeling.

- Conditional assignments in algorithmic requirements occasionally leave variable values unspecified. We assign default values to such cases in the formal specification when it is clear that the variable's value is a "don't care."

- Certain principal function inputs are treated by the environment as write-only variables to realize such things as signaling flags. After reading such a flag, the principal function resets it, which has an effect on the environment because the input is implemented via a global variable. The principal function output table will not show this variable as an output, however, because it is never read by the external environment. Nevertheless, to model this situation accurately a pseudo-output is provided in the formal specification's principal function interface to represent the effect of resetting the external variable.

- Similarly, the converse situation with principal function outputs that are treated as read-only by the environment can also arise. Such outputs are given corresponding pseudo-inputs to model any internal read accesses that may be required.

Chapter 6

GPS Formal Specifications

This chapter illustrates the formalization approach outlined in Chapter 5 using excerpts from the GPS formal specifications. Figures 6.1 through 6.6 show the selected excerpts. The full formal specifications used to model the requirements for the two GPS principal functions contain over 3200 lines of PVS notation (including comments and blank lines), packaged as eleven PVS theories. Appendix B contains a complete listing of these theories, as found in the latest revision of the GPS specifications. The PVS files are also available electronically via either FTP or WWW:

```
air16.larc.nasa.gov:pub/fm/larc/GPS-specs
http://atb-www.larc.nasa.gov/ftp/larc/GPS-specs
```

Another directory of interest is

```
http://atb-www.larc.nasa.gov/ftp/larc/PVS-library
```

which contains useful libraries of PVS files.

The specification version in Appendix B represents the baselined requirements that were approved for use by the Shuttle software development team. Earlier versions of the requirements and their corresponding PVS specifications were used to conduct the analyses described in Chapter 7.

6.1 Types and Common Operations

Figure 6.1 shows a portion of the vector and matrix utilities needed to formalize operations in this application domain. Using a parameterized theory such as this made it easy to declare vectors of reals where the index type differs from one vector type to the next. An alternative formulation would use the dimensionality, expressed as a natural number, as the theory parameter. Using a parameterized index type has the advantage of being more general at the expense of difficulty in expressing constraints derived from dimensionality. The one case where this arose was the cross product operation, which is only defined for vectors having dimension three or more.

Figure 6.2 illustrates the declaration of some typical types found in this application. All the types needed are rather simple and concrete; structured types are all of fixed size. As is customarily done in PVS, vectors and arrays are represented by function types. Notice how vectors of various kinds (e.g., `M50_vector`) are easily introduced by importing a parameterized theory.

```
vectors [index_type: TYPE]: THEORY
BEGIN

vector:          TYPE = [index_type -> real]

i,j,k:           VAR index_type
a,b,c:           VAR real
U,V:             VAR vector

zero_vector:         vector = (LAMBDA i: 0)
vector_sum(U, V):    vector = (LAMBDA i: U(i) + V(i))
vector_diff(U, V):   vector = (LAMBDA i: U(i) - V(i))
scalar_mult(a, V):   vector = (LAMBDA i: a * V(i))

 . . .

END vectors
```

Figure 6.1: Vector operations organized as a PVS theory.

```
major_mode_code:     TYPE = nat
mission_time:        TYPE = real
GPS_id:              TYPE = {n: nat | 1 <= n & n <= 3}

receiver_mode:       TYPE = {init, test, nav, blank}
AIF_flag:            TYPE = {auto, inhibit, force}

M50_axis:            TYPE = {Xm, Ym, Zm}

IMPORTING            vectors[M50_axis]

M50_vector:          TYPE = vector[M50_axis]

position_vector:     TYPE = M50_vector
velocity_vector:     TYPE = M50_vector
GPS_positions:       TYPE = [GPS_id -> position_vector]
GPS_velocities:      TYPE = [GPS_id -> velocity_vector]

GPS_predicate:       TYPE = [GPS_id -> bool]
GPS_times:           TYPE = [GPS_id -> mission_time]
GPS_FOM_vector:      TYPE = [GPS_id -> GPS_figure_of_merit]
```

Figure 6.2: Selected type declarations.

6.2 Subfunctions

Figure 6.3 presents one of the subfunctions from GPS Receiver State Processing. The outputs are bundled together into a single record type and used as the result type for the PVS function used to model the Shuttle software subfunction. The definition of the PVS function contains a single expression, a record constructor that gives values for each of the required outputs, cast in the following form:

```
(# G_ref_anncd_reset      := <expression>,
   GPS_anncd_reset        := <expression>,
   GPS_anncd_reset_avail  := <expression>,
   R_ref_anncd_reset      := <expression>,
   T_anncd_reset          := <expression>,
   T_ref_anncd_reset      := <expression>,
   V_IMU_ref_anncd_reset  := <expression>,
   V_ref_anncd_reset      := <expression>
 #)
```

In this case, all the expressions are structured objects with GPS_id as the index type. Therefore, lambda-expressions with the variable I ranging over GPS_id are used to construct suitable values.

To further illustrate the approach, consider the following example:

```
(LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I) ELSE null_position ENDIF)
```

Because this term is a lambda-expression it evaluates to a function from $\{1, 2, 3\}$ to position vectors. For GPS receiver I, if its "data good" flag is set (GPS_DG_SF(I) holds), then the position value R_GPS(I) derived from the input R_GPS is the resulting value, otherwise a default position value is used.

In several cases, the subfunction requirements are fairly complex and it was necessary to introduce intermediate PVS functions to decompose the formalization. While this is a natural thing to do, it does cause some loss of traceability to the original requirements. Clarity and readability were judged more important, however, and such decompositions were introduced as needed. A consistent decomposition scheme helped make the use of such intermediates as transparent as possible.

6.3 Principal Functions

Figure 6.4 shows the method of modeling principal function interfaces as records of individual values corresponding to Shuttle program variables. Because the interfaces at this level are quite broad, some of these lists become moderately long, on the order of 20 or 30 elements. In reality, these inputs and outputs are not actually "passed" in any programming language sense during execution; they are usually accessed as global variables and thus can be thought of as having the semantics of "call by reference." Consequently, our formalization must necessarily be viewed as a model of the software structure, and in some cases there are unpleasant artifacts of the difference between the model and the real system.[1]

[1]For example, when output variables are to be updated conditionally, the requirements often fail to specify their values in the "else" cases. In practice, these variables would retain their previous values, while in the formal

```
ref_state_anncd_reset_out: TYPE = [#
      G_ref_anncd_reset:        GPS_accelerations,
      GPS_anncd_reset:          GPS_predicate,
      GPS_anncd_reset_avail:    GPS_predicate,
      R_ref_anncd_reset:        GPS_positions,
      T_anncd_reset:            GPS_times,
      T_ref_anncd_reset:        GPS_times,
      V_IMU_ref_anncd_reset:    GPS_velocities,
      V_ref_anncd_reset:        GPS_velocities
      #]

ref_state_announced_reset(DT_anncd_reset:  delta_time,
                          G_two:           GPS_accelerations,
                          GPS_DG_SF:       GPS_predicate,
                          GPS_SW_cap:      num_GPS,
                          R_GPS:           GPS_positions,
                          T_anncd_reset:   GPS_times,
                          T_current_filt:  mission_time,
                          T_GPS:           GPS_times,
                          V_current_GPS:   GPS_velocities,
                          V_GPS:           GPS_velocities
                          ) : ref_state_anncd_reset_out =

   (# G_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN G_two(I) ELSE null_acceleration ENDIF),
      GPS_anncd_reset       :=
        (LAMBDA I: GPS_DG_SF(I) AND
                   (T_current_filt - T_anncd_reset(I) > DT_anncd_reset)),
      GPS_anncd_reset_avail := GPS_DG_SF,
      R_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I) ELSE null_position ENDIF),
      T_anncd_reset         :=
        (LAMBDA I: IF GPS_DG_SF(I) AND
                      (T_current_filt - T_anncd_reset(I) > DT_anncd_reset)
                      THEN T_current_filt
                      ELSE null_mission_time
                   ENDIF),
      T_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN T_GPS(I) ELSE null_mission_time ENDIF),
      V_IMU_ref_anncd_reset :=
        (LAMBDA I: IF GPS_DG_SF(I)
                      THEN V_current_GPS(I)
                      ELSE null_velocity
                   ENDIF),
      V_ref_anncd_reset     :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN V_GPS(I) ELSE null_velocity ENDIF)
   #)
```

Figure 6.3: Sample subfunction of Receiver State Processing.

```
rec_sp_inputs: TYPE = [#
      crew_deselect_rcvr:     GPS_predicate,
      FOM:                    GPS_FOM_vector,
       . . .                   . . .
      V_GPS_ECEF:             GPS_velocities_WGS84,
      V_last_GPS:             GPS_velocities
      #]

rec_sp_state: TYPE = [#
      G_two_prev:             GPS_accelerations,
      GPS_DG_SF_prev:         GPS_predicate,
       . . .                   . . .
      V_last_GPS_sel:         velocity_vector,
      V_last_GPS_two:         velocity_vector
      #]

rec_sp_I_loads: TYPE = [#
      alt_SF:                 nonzero_real,
      ang_SF:                 nonzero_real,
       . . .                   . . .
      SF_vel:                 nonzero_real,
      sig_diag_GPS_nom:       cov_diagonal_vector
      #]

rec_sp_K_loads: TYPE = [#
      acc_prop_min:           real,
      GPS_SW_cap:             num_GPS
      #]

rec_sp_constants: TYPE = [#
      ATFL_OV:                nat,
      deg_to_rad:             real,
      nautmi_per_ft:          real
      #]

rec_sp_outputs: TYPE = [#
      corr_coeff_GPS:         corr_coeff_vector,
      crew_des_rcvr_rcvd:     GPS_predicate,
       . . .                   . . .
      V_IMU_ref_anncd_reset:  GPS_velocities,
      V_ref_anncd_reset:      GPS_velocities
      #]

rec_sp_result: TYPE = [# output: rec_sp_outputs, state: rec_sp_state #]
```

Figure 6.4: Principal function interface types.

Figures 6.5 and 6.6 depict the top-level structure of the GPS Receiver State Processing model. Its interface types are given by the declarations shown in Figure 6.4. Its body has the following form:

```
LET sf_1_out = subfun-1(...),
     . . .       . . .
    sf_n_out = subfun-n(...)
IN
   (# output := (# ... #),
      state  := (# ... #)
    #)
```

Each local variable assignment of the LET-expression represents the invocation of a subfunction and the storage of its intermediate results. Components of these local variables can be used directly as principal function outputs or passed to later subfunctions on the list. The final expression denotes the ultimate principal function result, which has the form of a record of output values plus state values. Each of these two is itself a record object having many components.

6.4 Organization and Statistics

PVS declarations must be grouped into theories. The nature of the GPS CR and the Shuttle software application suggest a theory organization centered on the principal function as the main architectural unit. Each principal function can then be presented in three PVS theories:

1. *Types and operations.* These would include unspecified functions to denote support operations appearing elsewhere in the Shuttle software hierarchy.

2. *Subfunctions.* Included would be each of the subfunctions modeled to whatever level of detail is considered appropriate.

3. *Principal function.* The top-level structure of the principal function, including all the interface variables, would appear here.

This is the basic organization we have followed, except that the subfunctions for Receiver State Processing were too complex to fit comfortably into a single theory, so that one theory was expanded into three.

At the next higher level of aggregation, the GPS specifications were organized into the following three groups of PVS theories:

1. *Common theories.* These consist of three short theories used to model vector and matrix operations needed to express GPS processing.

2. *GPS Receiver State Processing.* Five theories were used to represent this principal function and its types and subfunctions.

specification they would be equated with default values, as discussed in Section 5.5. As long as the variables in question are considered "dead" under the "else" conditions, there should be no serious consequences. One must always be aware of such modeling artifacts, however, to ensure that they cause no distortions during analysis.

```
GPS_receiver_state_processing(rec_sp_inputs:     rec_sp_inputs,
                              rec_sp_state:      rec_sp_state,
                              rec_sp_I_loads:    rec_sp_I_loads,
                              rec_sp_K_loads:    rec_sp_K_loads,
                              rec_sp_constants:  rec_sp_constants
                              ) : rec_sp_result =

  LET IMU_assign_out =
          IMU_assign(
              GPS_installed        (rec_sp_I_loads),
              GPS_SW_cap           (rec_sp_K_loads),
              nav_IMU_to_GPS       (rec_sp_inputs),
              V_current_filt       (rec_sp_inputs),
              V_last_GPS_two       (rec_sp_state) ),

      nav_state_prop_out =
          nav_state_propagation(
              G_two_prev           (rec_sp_state),
              GPS_DG               (rec_sp_inputs),
               . . .                . . .
              V_last_GPS_prev      (IMU_assign_out),
              V_last_GPS_sel       (rec_sp_state) ),

      SV_qual_assess_out =
          state_vector_quality_assessment(
              G_two                (nav_state_prop_out),
              GPS_DG_SF            (nav_state_prop_out),
               . . .                . . .
              V_GPS                (nav_state_prop_out),
              V_GPS_prev           (nav_state_prop_out) ),

      state_vect_sel_out =
          state_vector_selection(
              corr_coeff_GPS_nom   (rec_sp_I_loads),
              crew_deselect_rcvr   (rec_sp_inputs),
               . . .                . . .
              V_GPS                (nav_state_prop_out),
              V_GPS_sel            (nav_state_prop_out) ),

      ref_st_ann_reset_out =
          ref_state_announced_reset(
              DT_anncd_reset       (rec_sp_I_loads),
              G_two                (nav_state_prop_out),
               . . .                . . .
              V_current_GPS        (IMU_assign_out),
              V_GPS                (nav_state_prop_out) ),
```

Figure 6.5: Principal function specification.

```
    GPS_downlist_out =
        GPS_downlist_computation(
            alt_SF                (rec_sp_I_loads),
            ang_SF                (rec_sp_I_loads),
             . . .                 . . .
            T_GPS_sel             (state_vect_sel_out),
            V_GPS_sel             (state_vect_sel_out) )

IN (# output := (#
        corr_coeff_GPS     := corr_coeff_GPS     (state_vect_sel_out),
        crew_des_rcvr_rcvd := crew_des_rcvr_rcvd (state_vect_sel_out),
         . . .                 . . .              . . .
        V_IMU_ref_anncd_reset :=
                  V_IMU_ref_anncd_reset          (ref_st_ann_reset_out),
        V_ref_anncd_reset  := V_ref_anncd_reset  (ref_st_ann_reset_out)
        #),

      state := (#
        G_two_prev         := G_two_prev         (SV_qual_assess_out),
        GPS_DG_SF_prev     := GPS_DG_SF_prev     (SV_qual_assess_out),
         . . .                 . . .              . . .
        V_last_GPS_sel     := V_last_GPS_sel     (nav_state_prop_out),
        V_last_GPS_two     := V_last_GPS_two     (nav_state_prop_out)
        #)
    #)
```

Figure 6.6: Principal function specification (cont'd).

3. *GPS Reference State Processing.* Three theories were used to represent this principal function and its types and subfunctions.

The two principal functions had types in common that could have been factored out of the type theories as written. Future versions of the specifications may incorporate additional commonality along these lines.

Table 6.1 presents summary statistics on the PVS theories found in Appendix B. The figures in this table are raw line counts and character counts. No attempt has been made to subtract out blank lines and comments. Note that there is no generally accepted measure of formal specifications corresponding to the "lines of code" metric commonly used for programming language source code.

Theory Name	Line Count	Character Count
rec_sp_types	275	8278
rec_sp_subfuns_1	347	12872
rec_sp_subfuns_2	876	35282
rec_sp_subfuns_3	230	9773
rec_sp_pf	483	22591
Subtotals	2211	88796
ref_sp_types	118	3382
ref_sp_subfuns	541	21598
ref_sp_pf	285	12135
Subtotals	944	37115
vectors	50	1274
matrices	31	757
matrix_mult	18	353
Subtotals	99	2384
Totals	3254	128295

Table 6.1: Summary statistics for PVS theories.

Chapter 7

Analysis Results

Analysis results from the GPS case study are summarized below. The two main activities of constructing PVS specifications and identifying potential requirements issues are discussed. On balance, the outcome suggests a positive outlook for formal methods as a requirements analysis technique. Some Shuttle requirements analysts are optimistic about the potential impact of formal methods, while others in the Shuttle community became curious about the potential benefits of formalization and expressed interest in learning more. In particular, during requirements inspections some of the requirements authors from Draper Laboratory were asking how we uncovered so many issues.

7.1 Phasing of Specifications with CR Schedule

Initially, the relevant portions of the CR were analyzed to determine the basic structure of the principal functions and how they are decomposed into subfunctions. Based on this organization, a general approach for modeling the functions and expressing the formal specifications in PVS was devised. A preliminary working document on this prescribed technique for writing formal specifications for the GPS CR was drafted and formed the basis of Chapter 5.

Next, the interfaces of the principal functions and their subfunctions were carefully scrutinized. Particular emphasis was placed on being able to identify the types of all inputs and outputs, and to match up all the data flows that are implicit in the tabular format presented in the requirements. While conducting this analysis and preparing to write the formal specifications, various minor discrepancies were detected in the CR and these were reported to Loral (now Lockheed Martin) requirements analysts.

A set of preliminary formal specifications was developed for the principal functions known as GPS Receiver State Processing and GPS Reference State Processing, using the language of PVS. Assumptions were made as needed to overcome the discrepancies encountered. Enough detail was provided in the formal specifications to characterize the functions with high precision. In parallel with this activity, several requirements analysts had been learning formal methods and PVS and positioning themselves to carry out this work after the trial project was finished.

Formalization of the two principal functions in PVS has been completed and revised three times to keep up with requirements changes. Because of the breadth of this CR, convergence has been slow. Requirements changes have been frequent and extensive as the CR was worked through the review process. Our initial formal specification was based on a preliminary version of the CR, before the two-phase implementation plan (single-receiver followed by three-receiver)

was adopted. Subsequent versions were written to model the single-string GPS CR and its modifications.

PVS versions were written for Mod B, Mod D/E and Mod F/G of the CR. This activity took place over a four-month period. Each modification was followed by a requirements inspection, accompanied by various issues written against the corresponding version of the CR. Some, but not all, of the responses to these issues would be incorporated into the following modification. Significant requirements changes were still being inserted at each of these modification stages.

7.2 Issues Identified through Formal Methods

The formal modeling step demonstrated that it is not difficult to bring the precision of formalization to bear on the type of requirements we examined. Expressing the requirements in the language of an off-the-shelf verification methodology was straightforward. We found PVS effective for this purpose; we feel other formal languages would also fare well. The higher-order logic features of PVS proved to be useful, although not strictly necessary because of the concrete nature of the GPS requirements. A less expressive language, such as one based on first-order logic, could have been used with only a small loss of elegance.

This much was unsurprising. What was more of a pleasant discovery was the number of problems found in the requirements as a simple consequence of carrying out the formalization. While many have claimed this as a benefit of formal methods, we can offer another piece of anecdotal evidence to support it. All of the errors identified so far have been due to carrying the analysis only to the point of typechecking. It was also our intention to take up some theorem proving as well, but this has had to wait for the requirements themselves to reach a firmer state of convergence.

7.2.1 Types of Requirements Issues

Feedback from requirements analysts indicated our approach was helpful in detecting three classes of errors normally tracked by the Shuttle program:

- Type 4 — requirements do not meet CR author's intent.
- Type 6 — requirements not technically clear, understandable and maintainable.
- Type 9 — interfaces inconsistent.

An example of Type 4 errors encountered in the CR is omission due to conditionally updating variables. Suppose, for example, one branch of a conditional assigns several variables, leaving them unassigned on the other branch. The requirements author intends for the values to be "don't cares" in the other branch, but occasionally this is faulty because some variables such as flags need to be assigned in both cases. Similar problems encountered are those due to overlapping conditions, leading to ambiguity in the correct assignments to make.

Examples of Type 9 errors include numerous, minor cases of incomplete and inconsistent interfaces. Missing inputs and outputs from tables, mismatches across tables, inappropriate types, and incorrect names are all typical errors seen in the subfunction and principal function interfaces. Most are problems that could be avoided through greater use of automation in the requirements capture process.

Issue Severity	Mod B	Mod D/E	Mod F/G	Totals
High Major	1	0	0	1
Low Major	7	3	0	10
High Minor	19	40	6	65
Low Minor	8	0	2	10
Totals	35	43	8	86

Table 7.1: Summary of issues detected by formal methods.

7.2.2 Summary Statistics

All requirements issues detected during the formalization were passed on to Loral (now Lockheed Martin) representatives. Those deemed to be real issues, that is, not caused by the misunderstandings of an outsider, were then officially submitted on behalf of the formal methods analysis as ones to be addressed during the requirements inspections. Severity levels are attached to valid issues during the inspections. This allowed us to get "credit" for identifying problems and led to some rudimentary measurements on the effectiveness of formalization.

Table 7.1 summarizes a preliminary accounting of the issues identified during our analysis. The issues are broken out by severity level for the three inspections of the CR that took place. A grand total of 86 issues were submitted for the three inspections. Of these issues, 72 of the 86 were of Type 9 (interfaces inconsistent). The rest were primarily scattered among Type 4 (requirements do not meet CR author's intent) and Type 6 (requirements not technically clear, understandable and maintainable). Appendix A lists the issues summarized by Table 7.1.

The meaning of the severity categories used in Table 7.1 is as follows:

1. High major — Cannot implement requirement.
2. Low major — Requirement does not correctly reflect CR author's intent.
3. High minor — "Support" requirements are incorrect or confusing.
4. Low minor — Minor documentation changes.

As can be seen by these results, the added precision of formalization used early in the lifecycle can yield tangible benefits. While many of these issues could have been found with lighter-weight techniques, the use of formal specifications can detect them *and* leave open the option of deductive analysis later on. Thus, these results by themselves suggest a potential boost from the use of formal methods plus the promise of additional benefits if proving is ultimately attempted.

It is worth noting that most errors detected in the CR during the formalization exercise were not directly found by typechecking or other automated analysis activity, but were detected during the act of writing the specifications or during the review and preparation leading up to the writing step. Additional problems were found during the typechecking phase as well. When we reach the point of modeling higher level properties and carrying out proofs, we expect to see fewer errors still. This is consistent with general observations practitioners have made about inspections and reviews. Light-weight forms of analysis applied early detect more problems and detect them quickly, but they are usually superficial. As more powerful analysis methods are introduced, we find more subtle problems, but they tend to be less numerous.

Chapter 8

Assessment of Impact

The GPS FM Task represents an application of formal methods to requirements analysis on a substantial new capability to be added to the Shuttle flight software (FSW). The size of the CR for just the first phase of implementation (single GPS receiver) exceeds 1000 pages. Because of the size and number of subsystems involved, as indicated in Figure 4.1, this application is one of high complexity. While the GPS FM task was scoped to two principal functions, it was sufficiently broad to constitute a true test of formal methods within a large new Shuttle FSW development activity.

8.1 Comparison with Existing Requirements Analysis Process

Building formal specification models in PVS for the GPS Receiver State Processing and GPS Reference State Processing principal functions helped Shuttle Requirements Analysts to address the process deficiencies identified in Section 3.2.2. The technical approach to building a formal model described in Chapter 5 imposed a method for conducting requirements analysis on each principal function. Defining the inputs, outputs and state variables at the principal function as well as subfunction levels, combined with PVS type checking, helped our Shuttle FM RA team identify many interface issues. As can be seen in Appendix A, the overwhelming majority of issues submitted by the team were interface issues. Moreover, expressing function internal logic and algorithms helped the team identify some consistency and completeness issues in the CR requirements. While the building of a formal specification cannot be construed as a full methodology for requirements analysis, the process did allow our FM RA team to methodically investigate interfaces and logical conditions within the functions. The team's ability to surface issues was noticed by other Shuttle community members at the three major GPS CR requirements inspections held from May through September of 1995. Specifically, the CR authors at Charles Stark Draper Laboratory expressed interest in our team's application of formal methods to the GPS CR.

While currently unable to quantify a completeness measure for RA using formal methods, the team believes FM can help establish a greater degree of confidence in its completeness. Typechecking by PVS of constants, variables, and inputs and outputs across function definitions helped establish a high degree of confidence in the consistency and completeness of interfaces. Though not pursued during this task, the team also believes defining properties and proving them as indicated in Chapter 9 would build confidence in the consistency and completeness of logical formulations in the CR. Having proven properties in some areas of the CR might also

allow testing to focus in other areas.

A key advantage of using formal methods during requirements analysis is having a formal specification model as a product of the analysis. The formal model, along with assumptions and rationale defined during its development, serve to document the results of the analysis. Furthermore, the model provides a basis from which to continue analysis, particularly of further requirements changes. The model can be updated and typechecked, new properties can be established, and old properties re-established.

8.2 Benefits vs. Cost from a Shuttle Community Perspective

From a Shuttle Requirements Analyst's perspective, the major benefit of formal methods is in improving the requirements analysis process, as discussed in Section 8.1. A further potential benefit is the indication that FM helped to identify errors earlier in the process. Because of the complexity of the GPS CR, a series of inspections was necessary over a six month period to completely evaluate the CR. An analysis of FM issues after the first inspection revealed, however, that of 28 issues submitted, only four of them would have been found at that time by conventional means. Most certainly many of these errors would have been detected at subsequent inspections. But this analysis is a clear indication that formal methods added rigor to the process of requirements analysis so that errors could be more easily uncovered.

Our RA team for the GPS FM task found that building formal specification models helped considerably in focusing on the requirements for the GPS software subsystem. In contrast, the FSSR change pages contain a high degree of design detail as well as requirements because of the nature of the FSSRs themselves, tending to obscure the requirements. Therefore, formal specification models offer a major potential benefit from a Shuttle community perspective by serving as working requirements models. The models can serve as a means of communication between Lockheed Martin RAs and Shuttle community members on assessing contemplated requirements changes and evaluating pending changes.

On the the cost side, our Shuttle GPS RA team found training in formal methods is a prerequisite. Because of the background needed in mathematical logic to apply the formal methods approach to building models, some analysts may have difficulty in relating to these types of models. A novice team working on a complicated system such as the Shuttle GPS software subsystem may also have difficulty in building a formal model. We recommend that an expert in formal methods and tools be retained as either a consultant or member of the team. From the team's experience with the GPS CR, we recommend establishing *a priori* analysis objectives to be achieved with the formal methods model, as these objectives will drive how to formulate the model.

Our GPS experience seems to indicate that systems with either many logical decisions or a number of interacting functions are well suited to a formal methods approach. While there likely are other types of systems that are well suited, we also recognize there may be some systems, e.g., a system implementing a well-defined and well-understood numerical algorithm, that are not well suited. Nevertheless, our experience on the GPS pilot task as well as other Shuttle pilot tasks indicates there are many areas in the Shuttle software where formal methods would provide the long term benefits described above at a reasonable cost.

Chapter 9

Recommendations for Future Work

Due to the iterations needed for the GPS requirements to stabilize, only the basic formalization step and several revisions were carried out during the case study. Several logical continuations are possible from this point of departure. Suggested courses of action are indicated below.

9.1 Maintaining Formal Specifications

Given the work already performed to formalize the GPS requirements, the most straightforward continuation is simply to maintain the existing PVS specifications in the face of regular updates. This should involve only minimal effort, assuming that the requirements do not undergo significant change. And if they do, the extra effort required to update the specifications may be well worth it if it leads to a more definitive evaluation of the impact of changes. The anticipated migration to the three-receiver GPS CR is an example of a moderate specification update that should contribute to an assessment of the full-up CR.

With a formalized version of the baselined requirements available, several benefits immediately accrue. By maintaining current PVS specifications of the CR, many effects of requirements changes can be evaluated directly, especially during early stages of software design before an implementation is available. Consider, for example, a requirements change that involves deleting several interface variables and adding several others. Reflecting those changes in the PVS specifications and typechecking again can be carried out quickly by analysts familiar with the PVS formulation. Immediately detectable would be several classes of errors such as omissions from interface tables, type mismatches, inappropriate operations on new variables, and processing steps still referring to deleted variables.

Another benefit is simply having a machine-readable form of the requirements at hand. Many exploratory questions can be answered simply by searching the PVS specification files for occurrences of any identifiers of interest. Auxiliary information about the requirements can be constructed from these specifications and maintained as well. Such information can include metrics about various aspects of the requirements and their interrelationships. Indexes and cross references are also readily constructed from the PVS theories, some of which are already available from the built-in PVS commands.

Finally, we expect the formalized requirements to be useful during system testing. The PVS theories could play a role in resolving requirements ambiguities that may arise during testing, either during the selection of test cases or in the interpretation of anomalous test results. While the requirements as expressed in PVS specifications may not turn out to be the ones originally

intended, their more precise expression in PVS should aid in the search for the cause of failed test cases. Testing may be further focused if various system properties can be established, as elaborated in the next section.

9.2 Formulating High-Level Properties

In addition to specifications that capture the functionality of principal functions, often it is desirable to formalize abstract properties about the long-term behavior of software subsystems. Formulating such properties is a way of assuring that certain critical constraints on system operation are always observed. It allows us to reason in a "longitudinal" manner by expressing what should be true about software behavior over time rather than merely what holds at the current step. The specification framework sketched here can be extended easily to accommodate property-oriented assertions. Coupling this technique with theorem proving would provide a powerful means of analyzing the requirements for adherence to key system-level operating goals.

A logical next step in the application of formal methods to GPS would be to identify and formalize important behavioral properties of the processing of GPS position and velocity vectors. In particular, the feedback loop shown in Figure 4.1 involving the principal functions Receiver State Processing and Reference State Processing is fertile ground for investigation. Proving that suitable properties hold would offer a powerful means of further shaking out the requirements.

Expressing behavioral properties and constraints in PVS is readily accomplished using the specification language features. Let us consider one possible method based on the following declarations for the *trace* concept:

```
trace_record: TYPE =
     [# input: pf_inputs, output: pf_outputs, state: pf_state #]

trace: TYPE = [nat -> trace_record]
```

This kind of trace is an infinite sequence of records $< H_0, H_1, \ldots >$ representing one possible history of events for the principal function in question. Formulation as a finite sequence is possible as well. The value H_i records the output and next state produced by the principal function during its ith execution, given that it receives the input found in H_i and has the previous state found in H_{i-1}. H_0 is used to specify the initial state.

Well-formedness conditions on traces are easily constructed:

```
valid_trace(H: trace, PF: principal_function): bool =
      valid_initial_state(state(H(0))) AND
      (FORALL i: LET step = PF(input(H(i+1)), state(H(i)),
                               I_loads, K_loads, constants)
                 IN output(H(i+1)) = output(step) AND
                    state(H(i+1))  = state(step) )
```

It may also be necessary to impose constraints on inputs, state values, and constants to characterize well-formedness. Such conditions are simply added to the `valid_trace` predicate.

Now properties may be expressed as predicates over traces:

```
(FORALL (H: trace): valid_trace(H, PF) => property(H))
```

Properties expressed in this way could take one of several forms depending on what type of relationship among trace records is desired:

1. `(FORALL i: P(H(i)))`

   This form imposes the predicate `P` on each trace record in isolation. This would be appropriate for specifying an invariant of the principal function's operation. Each output-state pair must satisfy the condition without reference to any earlier trace elements. A typical use would be expressing how important variables should be related to one another. Flag variables, for example, often need to satisfy mutual exclusion and this would be a good place to express that constraint.

2. `(FORALL i: P(H(i)) => Q(H(i), H(i+1)))`

   A second form compares successive pairs of trace records. This might be used to characterize valid state transitions, sequencing properties, proper reaction to transients and mode changes, and plausibility checks on elaborate computations. Here the predicate `P` is used to select when `Q` should apply, or alternatively, to rule out comparisons at the beginning and at transitions that need to be exempt from the constraints.

3. `(FORALL i: P(H, i))`

   A third form is used to compare the current trace record to the entire sequence of records that preceded it. `P` typically would be defined as a function recursive in the variable `i`. Uses for this form include expressing comprehensive sequencing requirements, cumulative execution effects, and desired cyclic behavior.

Clearly, other forms of property formulation are possible. Note that properties such as these are usually proved with some form of mathematical induction. It is often necessary to state a more comprehensive property than desired so that the induction hypothesis is strong enough to complete the proof.

Given the preceding definitions, consider an example to illustrate the formulation of properties. Suppose we wish to express a feasibility criterion on selected GPS state vectors. In particular, suppose we want to ensure that a newly selected state vector does not deviate too far from the one previously selected. The physics of Shuttle flight dictate how much change in position and velocity are possible over a short time interval. If we just stipulate that differences between adjacent state vectors are within suitable upper bounds, then we can use the resulting property to establish that no wildly errant state vectors will be transmitted by the GPS subsystem.

Figure 9.1 shows how we might formulate this idea in PVS. A few simplifications have been assumed for this illustration, namely, that a selected state vector is available each time, and no conditions exist that might disqualify a state vector from consideration. In actual use, such conditions would need to be factored into the predicates.

9.3 Opportunities for Deductive Analysis

Expressing high-level properties of the form suggested in the previous section offers another valuable means of analyzing requirements. Just as with the specification of functionality itself,

```
valid_pair(H1, H2: trace_record): bool =
      feasible_delta(sel_state(output(H1)), sel_state(output(H2)))

feasible_delta(SV1, SV2: state_vector): bool =
      time(SV2) - time(SV1) <= near_time_limit IMPLIES
        vector_mag(vector_diff(position(SV2), position(SV1)))
          <= max_position_delta  AND
        vector_mag(vector_diff(velocity(SV2), velocity(SV1)))
          <= max_velocity_delta
```

Figure 9.1: Sample behavioral property for GPS Receiver State Processing.

the act of formalizing behavioral properties can often lead to the detection of errors without even attempting to prove them. Formulating invariants and other constraints forces attention on certain aspects of system behavior and how they must be realized. By writing them down, one often notices cases that may not be addressed properly or areas of processing that look insufficiently elaborated.

Nevertheless, the most benefit to be obtained from formal methods comes from the proof of important system properties. Constructing a proof leads to additional scrutiny being placed on the area under study. By shining the strong light of mechanical theorem proving in the dark corners of a specification, it is possible to reveal flaws that might escape detection even by extensive test procedures. Conversely, once the proof of a property has been established, high confidence in that aspect of system behavior is justified based on the strong evidence of logical deduction.

PVS provides a powerful mechanical prover for carrying out such deductions. Using it successfully requires more experience than needed to write specifications. Once a comfortable level of proficiency in specification writing has been achieved, further analyses based on proof techniques may be explored. It is best to start with modest undertakings and proceed gradually to more ambitious proof attempts. Effective prover use requires a good understanding of how to attack the proof activity methodically and decompose the effort into manageable pieces.

One aspect of using a prover that may not be apparent is the need to carefully structure formal specifications to facilitate proving. Prover performance is sensitive to the complexity of specification expressions. While the typechecker may be able to cope with a certain level of complexity, the prover's performance can degrade significantly when presented with expressions containing many terms. The solution is to introduce intermediate function definitions and supporting lemmas that make deduction steps small enough that the prover is not trying to handle too many details at once.

The GPS specifications developed during the case study have not been organized to enhance theorem proving. Emphasis was placed on fidelity to the CR requirements. Repackaging of the specifications will probably be necessary before embarking on an extensive proof campaign. The one area likely to need attention is the use of long lists of inputs for the various specification functions. Some grouping of the inputs using record types may be advisable so fewer separate arguments appear. A similar grouping of output and state variable lists may also be helpful. These are already specified using record types, but additional grouping into hierarchies of records should be considered. This would reduce the number of components in the large record types and improve the prover's performance during simplification steps. Experimental

modifications such as these may be done gradually to determine if and to what extent they are needed.

Chapter 10

Conclusions

Experience with the GPS effort showed that the outlook for formal methods in this requirements analysis domain is quite promising. PVS has been used effectively to formalize this application, and the state machine specification approach should be easy to duplicate for other areas. There are good prospects for continuation of the effort by Lockheed Martin personnel. Some Shuttle RAs are optimistic about the potential impact of FM. Although the specification activity was assisted by tools, doing manual specification is also feasible here, albeit with reduced benefits.

PVS provides a formal specification language of considerable theoretical power while still preserving the syntactic flavor of modern programming languages. This makes the specifications fairly readable to nonexperts and makes their development less difficult than might otherwise be the case with specification languages whose features are more limiting. The scheme detailed here leads to specifications that RAs and others from the Shuttle community can and did learn to read and interpret without having to become PVS practitioners. Moreover, the mere construction of formal specifications using this method can and did lead to the discovery of flaws in the requirements. Future efforts can use the specifications as the foundation for more sophisticated analyses based on the use of formal proof. This additional tool provides the means to answer nontrivial questions about the specifications and achieve a higher level of assurance that the requirements are free of major flaws.

An additional (longer term) possibility is for the requirements authors themselves to use formal specifications in writing the original requirements. This approach would offer the advantage of more rigor from the outset, eliminating the more trivial requirements issues identified in this study. Requirements analysts could then concentrate fully on more substantial issues, making better use of their time and effort. Using their time to conduct deductive analyses, for example, becomes more feasible under this scenario.

The methods outlined for formally specifying requirements were devised to meet the needs of the chosen CR. They are methods having fundamental utility that should lend themselves to other avionics applications. Tailoring a scheme for other uses or fine tuning it for the intended CR is easily accomplished. Alternative specification styles could readily be adopted. Experience in using the methods on real-world applications will help determine what direction future refinements should take.

Finally, it is worth considering how formalizing requirements would help overcome the deficiencies cited earlier for the current requirements analysis process:

1. *There is no methodology to guide the analysis.*

Formal methods offer rigorous modeling and analysis techniques that bring increased precision and error detection to the realm of requirements.

2. *There are no completion criteria.*

   Writing formal specifications and conducting proofs are deliberate acts to which one can attach meaningful completion criteria.

3. *There is no structured way for RAs to document the results of their analysis.*

   Formal specifications are tangible products that can be maintained and consulted as analysis and development proceed. When provided as outputs of the analysis process, formalized requirements can be used as evidence of thoroughness and coverage, as definitive explanations of how CRs achieve their objectives, and as permanent artifacts useful for answering future questions and addressing future changes.

While other methods could address these deficiencies to varying degrees, the theoretical power of mathematical models justifies the initial investment needed to create them.

Bibliography

[1] Ricky W. Butler, James L. Caldwell, Victor A. Carreno, C. Michael Holloway, Paul S. Miner, and Ben L. Di Vito. NASA Langley's research and technology transfer program in formal methods. In *Tenth Annual Conference on Computer Assurance (COMPASS 95)*, Gaithersburg, MD, June 1995.

[2] Judith Crow and Ben L. Di Vito. Formalizing space shuttle software requirements. In *Workshop on Formal Methods in Software Practice (FMSP '96)*, pages 40–48, San Diego, California, January 1996.

[3] Judy Crow. Finite-State Analysis of Space Shuttle Contingency Guidance Requirements. Technical Report SRI-CSL-95-17, Computer Science Laboratory, SRI International, Menlo Park, California, December 1995. Also published as NASA Contractor Report 4741, May 1996.

[4] Ben L. Di Vito. Formalizing new navigation requirements for NASA's space shuttle. In *Formal Methods Europe (FME '96)*, pages 160–178, Oxford, England, March 1996. Also Lecture Notes in Computer Science 1051, Springer.

[5] David Hamilton, Rick Covington, and John Kelly. Experiences in applying formal methods to the analysis of software and system requirements. In *WIFT '95: Workshop on Industrial-Strength Formal Specification Techniques*, pages 30–43, Boca Raton, FL, 1995. IEEE Computer Society.

[6] Robyn R. Lutz and Yoko Ampo. Experience Report: Using Formal Methods for Requirements Analysis of Critical Spacecraft Software. In *19th Annual Software Engineering Workshop*, pages 231–248. NASA GSFC, 1994. Greenbelt, MD.

[7] Multi-Center NASA Team from Jet Propulsion Laboratory, Johnson Space Center, and Langley Research Center. *Formal Methods Demonstration Project for Space Applications - Phase I Case Study: Space Shuttle Orbit DAP Jet Select*, December 1993. NASA Code Q Final Report.

[8] National Aeronautics and Space Administration, Office of Safety and Mission Assurance, Washington, DC. *Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion*, July 1995.

[9] National Research Council Committee for Review of Oversight Mechanisms for Space Shuttle Flight Software Processes, National Academy Press, Washington, DC. *An Assessment of Space Shuttle Flight Software Development Practices*, 1993.

[10] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[11] John Rushby. Formal methods and digital systems validation for airborne systems. NASA Contractor Report 4551, December 1993.

[12] John Rushby. Formal methods and their role in digital systems validation for airborne systems. NASA Contractor Report 4673, NASA, August 1995.

[13] A. John van Schouwen. The A-7 Requirements Model: Re-Examination for Real-Time Systems and an Application to Monitoring Systems. Technical Report 90-276, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, May 1990.

Appendix A

Issues Uncovered

The following is a list of the requirements issues surfaced during the GPS case study. These issues were submitted during the official requirements inspection process and have been used to revise and correct the actual GPS CR. A summary of the issues and a discussion of their significance appear in Chapter 7. Note that the issue titles in the following list were truncated by the database report generator when the list was created.

ISSUE TRACKING REPORT 11/16/95

SEARCH FOR IDENTIFIER=91051, SOURCE=FORMAL
SORT BY ISSUE TYPE, ISSUE SEVERITY,

TRACKING IDENTIFIER	TITLE	ISSUE NUM	ISSUE SOURCE	ISSUE SEVERITY	PF	ISSUE TYPE
91051B	NO VARIABLE CROSS REFERENCE	049	FORMAL MTH	HMINOR	4.21	3
91051B	GPS_RCVR_STATUS TESTED INST	345	FORMAL MTH	HMAJOR	4.18	4
91051B	'OR' IN CONDITIONAL STATEME	340	FORMAL MTH	LMAJOR	4.18	4
91051B	GPS_RCVR_ANNCD_RESET TESTED	358	FORMAL MTH	LMAJOR	4.18	4
91051B	LAST SENTENCE OF PARAGRAPH	617	FORMAL MET	LMAJOR	4.18	4
91051B	FOM_SV_SEL	053	FORMAL MTH	LMAJOR	4.21	4
91051B	GPS_RCVR_ANNCD_RESET TEST U	359	FORMAL MTH	LMINOR	4.18	4
91051G	R_GPS_PREV, V_GPS_PREV INPU	334	FORMAL MET	HMINOR	4.21	6
91051E	NAUTMI_OER_FT INCORRECTLY R	209	FORMAL MTH	HMINOR	4.21	6
91051E	DELTA_RATIO_SV_SEL_STAT INC	206	FORMAL MTH	HMINOR	4.21	6
91051E	GPS_DELV_QA3(I) TESTED AGAI	191	FORMAL MTH	LMAJOR	4.21	6
91051E	QA_OVERRIDE INCORRECTLY REF	182	FORMAL MTH	LMAJOR	4.21	6
91051E	NUM_GPS_SEL TESTED AGAINST	183	FORMAL MTH	LMAJOR	4.21	6
91051B	TABLE 4.3.3.5-1 CHANGES	054	FORMAL MTH	HMINOR	4.21	9
91051B	INCORRECT OUTPUT DESTINATIO	052	FORMAL MTH	HMINOR	4.21	9
91051B	ADD CONSTANT GO TO TABLE 4.	051	FORMAL MTH	HMINOR	4.21	9
91051B	TABLE 4.3.3.2-1 SOURCE OF V	048	FORMAL MTH	HMINOR	4.21	9
91051B	FILT_RESTART_DISPLAY IS AN	616	FORMAL MET	HMINOR	4.18	9
91051B	GPS_INIT_REQUEST SHOULD BE	361	FORMAL MTH	HMINOR	4.18	9
91051B	GPS_MODE NOT NEEDED IN TABL	360	FORMAL MTH	HMINOR	4.18	9
91051B	R_AVGG AND V_AVGG MISSING F	354	FORMAL MTH	HMINOR	4.18	9
91051B	G_GPS HAS EXTRA SOURCE IN T	353	FORMAL MTH	HMINOR	4.18	9

91051B	T_GPS_OUTPUT_BIAS SHOULD BE	352	FORMAL MTH	HMINOR	4.18	9
91051B	GPS_INSTALLED IS MISSING FR	351	FORMAL MTH	HMINOR	4.18	9
91051B	FILT_RESTART IS MISSING FRO	350	FORMAL MTH	HMINOR	4.18	9
91051B	GM_DEG_GPS_AID IS MISSING F	349	FORMAL MTH	HMINOR	4.18	9
91051B	GM_ORD_GPS_AID IS MISSING F	348	FORMAL MTH	HMINOR	4.18	9
91051B	GPS_ANNCD_RESET IS MISSING	347	FORMAL MTH	HMINOR	4.18	9
91051B	UNNEEDED INPUTS IN TABLE 4.	343	FORMAL MTH	HMINOR	4.18	9
91051B	INCONSISTENT DESTINATIONS	342	FORMAL MTH	HMINOR	4.18	9
91051B	GPS_INSTALLED MISSING FROM	341	FORMAL MTH	HMINOR	4.18	9
91051G	GPS_SEL_DELR(DELV)_UVW PARA	344	FORMAL MET	HMINOR	4.21	9
91051G	MISSING GPS SV SELECT INPUT	343	FORMAL MET	HMINOR	4.21	9
91051G	GPS_QA3_RTOL, VTOL INPUT PA	337	FORMAL MET	HMINOR	4.21	9
91051G	DT_QA2 PARAMETER	332	FORMAL MET	HMINOR	4.21	9
91051G	DELTA_RATIO_QA2_IND, MAX OU	331	FORMAL MET	HMINOR	4.21	9
91051E	INCORRECT INDEXING FOR GPS_	236	FORMAL MTH	HMINOR	4.21	9
91051E	MISSING INPUTS FOR TABLE 4.	233	FORMAL MTH	HMINOR	4.21	9
91051E	DELR NOT NEEDED IN TABLE	232	FORMAL MTH	HMINOR	4.21	9
91051E	DELR_RATIO_QA2_DISPLAY IS N	231	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR CORR_C	230	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR GPS_AL	229	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR GPS_SV	228	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR SIG_DI	227	FORMAL MTH	HMINOR	4.21	9
91051E	T_GPS_LAST_UPDATE MISSING F	217	FORMAL MTH	HMINOR	4.21	9
91051E	DELR_RATIO_SV_SEL DOES NOT	226	FORMAL MTH	HMINOR	4.21	9
91051E	DELTA_RATIO_IND_SEL DOES NO	225	FORMAL MTH	HMINOR	4.21	9
91051E	GPS_LAT/LON_IND HAS DIFFERE	224	FORMAL MTH	HMINOR	4.21	9
91051E	MVS_RCVR_SEL NOT ON PF OUTP	223	FORMAL MTH	HMINOR	4.21	9
91051E	MISSING OUTPUTS FROM TABLE	222	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR GPS_DG	221	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT COLUMN HEADING	248	FORMAL MTH	HMINOR	4.21	9
91051E	CREW_DES_RCVR IMPROPERLY RE	247	FORMAL MTH	HMINOR	4.21	9
91051E	PRINCIPAL FUNCTION OUTPUT T	237	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE FOR GPS_DG_	246	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE FOR GPS_OFF	245	FORMAL MTH	HMINOR	4.21	9
91051E	UNKNOWN SOURCE FOR GPS_TO_N	244	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE FOR MVS_RCV	243	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE FOR SF_POS	242	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE AND REFEREN	241	FORMAL MTH	HMINOR	4.21	9
91051E	MISSING INPUTS FOR TABLE 4.	240	FORMAL MTH	HMINOR	4.21	9
91051E	GPS_COLLECTION_WORD INCORRE	239	FORMAL MTH	HMINOR	4.21	9
91051E	R/V_GPS_SEL_DL IMPROPERLY L	238	FORMAL MTH	HMINOR	4.21	9
91051E	DA_THRESHOLD MISSING FROM K	168	FORMAL MTH	HMINOR	4.21	9
91051E	IMPROPER SOURCE FOR G_TWO_P	165	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR R/T/V_	164	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR V_LAST	163	FORMAL MTH	HMINOR	4.21	9
91051E	R/V_GPS AND R_GPS_PREV NOT	162	FORMAL MTH	HMINOR	4.21	9
91051E	G_TWO MISSING FROM INPUT TA	216	FORMAL MTH	HMINOR	4.21	9
91051E	OUTPUTS DELR/DELV_RATIO_QA3	214	FORMAL MTH	HMINOR	4.21	9
91051E	DELR_MAG MISSING AS AN INTE	213	FORMAL MTH	HMINOR	4.21	9
91051E	GPS DG_SF_PREV LISTED TWICE	212	FORMAL MTH	HMINOR	4.21	9
91051E	INCORRECT SOURCE FOR GPS_DG	210	FORMAL MTH	HMINOR	4.21	9
91051E	ACC_PROP_MIN IMPROPERLY REF	169	FORMAL MTH	HMINOR	4.21	9
91051B	GPS_MODE NOT CONSISTENTLY N	339	FORMAL MTH	LMAJOR	4.18	9
91051B	GPS_INIT_REQUEST NOT CONSIS	338	FORMAL MTH	LMAJOR	4.18	9
91051G	IMU_1(2,3)_FAIL PARAMETERS	156	FORMAL MET	LMINOR	4.18	9
91051G	IMU_SFC_GPS	157	FORMAL MET	LMINOR	4.18	9
91051B	GPS_REF_INIT IS MISSING FRO	363	FORMAL MTH	LMINOR	4.18	9

91051B	GPS_DEG_GPS_AID IS MISSING	362	FORMAL MTH	LMINOR	4.18	9
91051B	GPS_SW_CAP IS LISTED TWOCE	357	FORMAL MTH	LMINOR	4.18	9
91051B	CLOCK/CLOCKTIME IS AN UNUSE	356	FORMAL MTH	LMINOR	4.18	9
91051B	T_CURRENT IS MISSING FROM T	355	FORMAL MTH	LMINOR	4.18	9
91051B	GPS_ORD_GPS_AID IS MISSING	346	FORMAL MTH	LMINOR	4.18	9
91051B	UNUSED OUTPUT IN TABLE 4.6.	344	FORMAL MTH	LMINOR	4.18	9
91051B	GPS RCVR STATE PROC I-LOADS	050	FORMAL MTH	LMAJOR	4.21	19

EXPLANATIONS:

PRINCIPAL FUNCTION (PF)
- 4.18 - GPS REFERENCE STATE PROCESSING
- 4.21 - GPS RECEIVER STATE PROCESSING

ISSUE SEVERITY
- HMAJOR - LORAL DOES NOT KNOW HOW TO IMPLEMENT THE REQUIREMENT
- HMINOR - EXISTING REQUIREMENT DOES NOT CORRECTLY REFLECT THE CR AUTHOR'S INTENT
- LMAJOR - "SUPPORT" REQUIREMENTS ARE INCORRECT OR CONFUSING
- LMINOR - MINOR DOCUMENTATION CHANGES

ISSUE TYPE
- 3 - ALL NECESSARY REQUIREMENTS PAGES INCLUDED
- 4 - REQUIREMENTS MEET CR AUTHOR'S INTENT; CR WILL WORK
- 6 - REQUIREMENTS TECHNICALLY CLEAR, UNDERSTANDABLE AND MAINTAINABLE
- 9 - INTERFACES DOCUMENTED AND CONSISTENT
- 19 - ILOAD, KLOAD AND LEVEL C DATA REQUIREMENTS COMMPLETE

Appendix B

Formal Specifications of the GPS Principal Functions

The full text of the PVS theories used to formalize the chosen subset of the GPS CR is contained in this appendix. Summary statistics for the PVS theories appear in Table 6.1 on page 33. The revision level of the requirements captured by these specifications is Mod K. Earlier versions of the requirements were used in the formalizations and analyses described in Chapter 7. The Mod K revision of the CR was baselined as the approved set of requirements used to begin software development.

PVS allows theories to be grouped together into files as long as no theory is split across more than one file. The eleven PVS theories of the GPS specifications are grouped into seven files. Their organization and starting page numbers within the listing are shown below.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% vector.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Thu Jun  8 09:35:49 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Fri May 17 18:34:41 1996
%% Update Count    : 9
%% Status          : Preliminary
%%
%% HISTORY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generic vector and matrix operations, currently limited to real elements.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


vectors [index_type: TYPE]: THEORY
BEGIN

vector:         TYPE = [index_type -> real]

i,j,k:          VAR index_type
a,b,c:          VAR real

U,V:            VAR vector

zero_vector:        vector = (LAMBDA i: 0)

vector_sum(U, V):  vector = (LAMBDA i: U(i) + V(i))

vector_diff(U, V): vector = (LAMBDA i: U(i) - V(i))

scalar_mult(a, V): vector = (LAMBDA i: a * V(i))

max(V): real
min(V): real

sum(V): real

dot_product(U, V): real =   sum((LAMBDA i: U(i) * V(i)))

sqrt(a): real

vector_mag(V): real =  sqrt(sum((LAMBDA i: V(i) * V(i))))

cross_product(U, V): vector          %% need to restrict to 3D

END vectors
```

```
matrices [row_type, col_type: TYPE]: THEORY
BEGIN

vector:         TYPE = [col_type -> real]
matrix:         TYPE = [row_type -> vector]

i:              VAR row_type
j:              VAR col_type
a,b,c:          VAR real

U,V:            VAR vector
M,N:            VAR matrix

vector_2:       TYPE = [row_type -> real]
matrix_2:       TYPE = [col_type -> vector_2]

IMPORTING vectors[col_type], vectors[row_type]

matrix_sum(M, N):  matrix = (LAMBDA i: vector_sum(M(i), N(i)))

matrix_diff(M, N): matrix = (LAMBDA i: vector_diff(M(i), N(i)))

scalar_mult(a, M): matrix = (LAMBDA i: (LAMBDA j: a * M(i)(j)))

transpose(M):      matrix_2 = (LAMBDA j: (LAMBDA i: M(i)(j)))

matrix_vector_mult(M, V): vector_2 = (LAMBDA i: dot_product(M(i), V))

END matrices


matrix_mult [T1, T2, T3: TYPE]: THEORY
BEGIN

IMPORTING matrices[T1, T2], matrices[T2, T3]

i:              VAR T1
j:              VAR T3
a,b,c:          VAR real

M:              VAR matrix[T1,T2]
N:              VAR matrix[T2,T3]

matrix_mult(M, N): matrix[T1,T3] =
       (LAMBDA i: (LAMBDA j: dot_product(M(i), transpose(N)(j))))

END matrix_mult
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rec_sp_types.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:21:12 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Tue May 14 09:28:49 1996
%% Update Count    : 128
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%       for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued 3/20/95
%%   Updated to comply with     GPS CR 91051D, issued 6/09/95
%%   Updated to comply with     GPS CR 91051E, issued 6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Receiver State Processing (4.21) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rec_sp_types: THEORY
BEGIN

%%%%%%%%
%% NOTE: PVS Boolean types are used to model Shuttle discrete types, whose
%%       constants are ON and OFF, as well as integer types restricted to the
%%       values 0 and 1.
%%%%%%%%


%%%%%%%%%%% Time types %%%%%%%%%%

mission_time:          TYPE = real

delta_time:            TYPE = real


%%%%%%%%%%% Index types %%%%%%%%%%

num_GPS:               TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

IMU_id:                TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

GPS_id:                TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

IMU_to_GPS_map:        TYPE = [IMU_id -> GPS_id]

corr_coeff_index:      TYPE = {n: nat | 1 <= n & n <= 7} CONTAINING 1

cov_position_index:    TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1
cov_velocity_index:    TYPE = {n: nat | 4 <= n & n <= 6} CONTAINING 4
cov_diagonal_index:    TYPE = {n: nat | 1 <= n & n <= 6} CONTAINING 1
cov_SF_table_index:    TYPE = {n: nat | 1 <= n & n <= 2} CONTAINING 1


%%%%%%%%%%% Coordinate systems and position/velocity vectors %%%%%%%%%%

M50_axis:              TYPE = {Xm, Ym, Zm}

IMPORTING              vectors[M50_axis]

M50_vector:            TYPE = vector[M50_axis]

position_vector:       TYPE = M50_vector
velocity_vector:       TYPE = M50_vector
acceleration_vector:   TYPE = M50_vector

delta_position_vector: TYPE = M50_vector
delta_velocity_vector: TYPE = M50_vector

GPS_positions:         TYPE = [GPS_id -> position_vector]
GPS_velocities:        TYPE = [GPS_id -> velocity_vector]
GPS_accelerations:     TYPE = [GPS_id -> acceleration_vector]

GPS_delta_positions:   TYPE = [GPS_id -> delta_position_vector]
GPS_delta_velocities:  TYPE = [GPS_id -> delta_velocity_vector]


EF_axis:               TYPE = {Xg, Yg, Zg}

IMPORTING              vectors[EF_axis]

EF_vector:             TYPE = vector[EF_axis]


WGS84_axis:            TYPE = {Xwgs, Ywgs, Zwgs}

IMPORTING              vectors[WGS84_axis]

WGS84_vector:          TYPE = vector[WGS84_axis]

position_vector_WGS84: TYPE = WGS84_vector
velocity_vector_WGS84: TYPE = WGS84_vector

GPS_positions_WGS84:   TYPE = [GPS_id -> position_vector_WGS84]
GPS_velocities_WGS84:  TYPE = [GPS_id -> velocity_vector_WGS84]
```

```
UVW_axis:              TYPE = {U, V, W}

IMPORTING              vectors[UVW_axis]

UVW_vector:            TYPE = vector[UVW_axis]

delta_position_vector_UVW: TYPE = UVW_vector
delta_velocity_vector_UVW: TYPE = UVW_vector

position_ratio_vector_UVW: TYPE = UVW_vector
velocity_ratio_vector_UVW: TYPE = UVW_vector

GPS_delta_pos_UVW:     TYPE = [GPS_id -> delta_position_vector_UVW]
GPS_delta_vel_UVW:     TYPE = [GPS_id -> delta_velocity_vector_UVW]


%%%%%%%%%% Modes and flags %%%%%%%%%%

AIF_flag:        TYPE = {auto, inhibit, force}

receiver_mode:   TYPE = {init, test, nav, blank}

major_mode_code: TYPE = nat     %% Fixed set of codes (not enumerated here)


%%%%%%%%%% Downlist types %%%%%%%%%%

collection_word:       TYPE+          %% Downlist collection word

M50_int_vector:        TYPE = [M50_axis -> int]
UVW_int_vector:        TYPE = [UVW_axis -> int]


%%%%%%%%%% Misc. types %%%%%%%%%%

GPS_figure_of_merit:  TYPE = {n: nat | 1 <= n & n <= 9} CONTAINING 1

distance:              TYPE = real

latitude:              TYPE = real
longitude:             TYPE = real

display_status:        TYPE+               %% Display status (char)
display_indicator:     TYPE+               %% Display indicator (char)

P_ind:                 display_indicator   %% position indicator 'P'
V_ind:                 display_indicator   %% velocity indicator 'V'
M_ind:                 display_indicator   %% direction indicator 'M'
S_ind:                 display_indicator   %% direction indicator 'S'
E_ind:                 display_indicator   %% direction indicator 'E'
W_ind:                 display_indicator   %% direction indicator 'W'

blank_stat:            display_status
down_arrow_stat:       display_status
up_arrow_stat:         display_status


%%%%%%%%%% Other GPS vectors %%%%%%%%%%

GPS_predicate:         TYPE = [GPS_id -> bool]

GPS_counters:          TYPE = [GPS_id -> nat]

GPS_times:             TYPE = [GPS_id -> mission_time]

GPS_ratios:            TYPE = [GPS_id -> real]

GPS_IMU_ids:           TYPE = [GPS_id -> IMU_id]

GPS_indicators:        TYPE = [GPS_id -> display_indicator]


GPS_rcvr_mode_vector: TYPE = [GPS_id -> receiver_mode]

GPS_FOM_vector:        TYPE = [GPS_id -> GPS_figure_of_merit]

tolerance_vector:      TYPE = UVW_vector

corr_coeff_vector:     TYPE = [corr_coeff_index -> real]

cov_diagonal_vector:   TYPE = [cov_diagonal_index -> real]


%% Following table structure not described in CR:

FOM_cov_SF_table:      TYPE = [cov_SF_table_index ->
                                   [GPS_figure_of_merit -> real]]


%%%%%%%%%% Coordinate conversion matrix types %%%%%%%%%%

IMPORTING              matrices[M50_axis, EF_axis],
                       matrices[EF_axis,  WGS84_axis],
                       matrices[M50_axis, WGS84_axis],
                       matrices[M50_axis, UVW_axis],
                       matrices[UVW_axis, M50_axis],
                       matrices[EF_axis, M50_axis]
```

```
EF_to_M50_matrix:     TYPE = matrix[M50_axis, EF_axis]
WGS84_to_EF_matrix:   TYPE = matrix[EF_axis,  WGS84_axis]
WGS84_to_M50_matrix:  TYPE = matrix[M50_axis, WGS84_axis]
UVW_to_M50_matrix:    TYPE = matrix[M50_axis, UVW_axis]
M50_to_UVW_matrix:    TYPE = matrix[UVW_axis, M50_axis]
M50_to_EF_matrix:     TYPE = matrix[EF_axis, M50_axis]


%%%%%%%%%% Default/null constants ("don't care" values) %%%%%%%%%%

null_mission_time:    mission_time
null_position:        position_vector
null_velocity:        velocity_vector
null_acceleration:    acceleration_vector

null_GPS_positions:   GPS_positions
null_GPS_velocities:  GPS_velocities


%%%%%%%%%% Misc. constants, types, and operations %%%%%%%%%%

truncate(x: real): int =
       IF x < 0 THEN ceiling(x) ELSE floor(x) ENDIF


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Supporting functions (from common NAV subfunctions)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%% Convert from earth-fixed to M50 coordinates %%%%%%%%%%

earth_fixed_to_M50_coord(T: mission_time) : EF_to_M50_matrix


%%%%%%%%%% Convert from UVW to M50 coordinates %%%%%%%%%%

UVW_to_M50(R: position_vector, V: velocity_vector) : UVW_to_M50_matrix


%%%%%%%%% Convert from earth-fixed coordinates to geodetic parameters %%%%%%%%%

EF_to_geodetic_out: TYPE = [#
      lat: latitude,
      lon: longitude,
      alt: distance
      #]

EF_to_geodetic(R: EF_vector): EF_to_geodetic_out


%%%%%%%%% Compute velocity relative to windless, rotating atmosphere %%%%%%%%%

V_rel(V: velocity_vector, R: position_vector): velocity_vector


%%%%%%%%%%% Super-G method for propagating state vectors %%%%%%%%%%%

super_G_out: TYPE = [#
      R_prop: position_vector,
      V_prop: velocity_vector,
      G_prop: acceleration_vector
      #]

super_G_GPS(R:      position_vector,
            V:      velocity_vector,
            DV:     delta_velocity_vector,
            DT:     delta_time,
            T:      mission_time,
            deg:    nat,
            ord:    nat,
            idrag:  nat,
            ivent:  nat) : super_G_out


END rec_sp_types
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rec_sp_subfuns_1.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:21:12 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Wed May 22 09:45:50 1996
%% Update Count    : 139
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%      for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued 3/20/95
%%   Updated to comply with     GPS CR 91051D, issued 6/09/95
%%   Updated to comply with     GPS CR 91051E, issued 6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Receiver State Processing (4.21) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%  Subfunctions, Group 1 %%%%%%%%%%%%%%%%%%%%%%%%%

rec_sp_subfuns_1: THEORY
BEGIN

IMPORTING rec_sp_types


%%%%%%%%%% Common variables %%%%%%%%%%

I:      VAR GPS_id


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.2: GPS IMU Assign (1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

IMU_assign_out: TYPE = [#
     V_current_GPS:   GPS_velocities,
     V_last_GPS_prev: GPS_velocities
     #]

IMU_assign(GPS_installed:     GPS_predicate,
           GPS_SW_cap:        num_GPS,
           nav_IMU_to_GPS:    GPS_IMU_ids,
           V_current_filt:    velocity_vector,
           V_last_GPS_two:    velocity_vector    %%% state var
           ) : IMU_assign_out =

   (# V_current_GPS :=
        (LAMBDA I: IF GPS_installed(I) AND nav_IMU_to_GPS(I) = 2
                     THEN V_current_filt
                     ELSE null_velocity
                   ENDIF),
      V_last_GPS_prev :=
        (LAMBDA I: IF GPS_installed(I) AND nav_IMU_to_GPS(I) = 2
                     THEN V_last_GPS_two
                     ELSE null_velocity
                   ENDIF)
    #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.3: GPS Navigation State Propagation (2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


super_G_prop: TYPE = [#
     R_prop: position_vector,
     V_prop: velocity_vector,
     G_prop: acceleration_vector,
     T_prop: mission_time
     #]

null_super_G_prop: super_G_prop

GM_params: TYPE = [# deg, ord:      nat,
                     DV:            delta_velocity_vector,
                     idrag, ivent:  nat #]

%%%%%%%%
%% NOTE: idrag_GPS and ivent_GPS may need to be treated as state
%%       variables.  They are assumed to be pure locals in the
%%       subfunction below.  When not assigned in the requirements,
%%       they are given values of zero.
%%%%%%%%

adjusted_DV(DV: delta_velocity_vector,
            thresh: real): delta_velocity_vector =
      IF vector_mag(DV) < thresh THEN zero_vector ELSE DV ENDIF

within_vel_thresh(DV: delta_velocity_vector, thresh: real): nat =
      IF vector_mag(DV) < thresh THEN 1 ELSE 0 ENDIF

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.3.1:  Propagation of the current valid receiver states
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

IMPORTING  matrix_mult[M50_axis, EF_axis, WGS84_axis]

integ_interval_thresh: real = 1/100000000   %% 1E-8

nav_state_prop_1(I:                  GPS_id,
                 GPS_IMU_vel_thresh: real,
                 M_WGS84_to_EF:      WGS84_to_EF_matrix,
                 nav_MM_code:        major_mode_code,
                 R_GPS_ECEF:         GPS_positions_WGS84,
                 T_current_filt:     mission_time,
                 T_GPS:              GPS_times,
                 T_last_GPS:         GPS_times,
                 V_current_GPS:      GPS_velocities,
                 V_GPS_ECEF:         GPS_velocities_WGS84,
                 V_last_GPS:         GPS_velocities) : super_G_prop =

  LET (M_WGS_to_M50: WGS84_to_M50_matrix) =
            matrix_mult(earth_fixed_to_M50_coord(T_GPS(I)), M_WGS84_to_EF),
      (R_GPS_I: position_vector) =
               matrix_vector_mult(M_WGS_to_M50, R_GPS_ECEF(I)),
      (V_GPS_I: velocity_vector) =
               V_rel(matrix_vector_mult(M_WGS_to_M50, V_GPS_ECEF(I)),
                     scalar_mult(-1, R_GPS_I)),

      DT =  T_current_filt - T_GPS(I),
      DT2 = IF DT < integ_interval_thresh THEN 0 ELSE DT ENDIF,

      DV =  scalar_mult(DT / (T_current_filt - T_last_GPS(I)),
                        vector_diff(V_current_GPS(I), V_last_GPS(I))),

      (GM_params: GM_params) =
         IF DT < integ_interval_thresh
            THEN  (# deg := 0, ord := 0,        %% Only DV specified in this
                     DV := zero_vector,         %% case; others assumed 0.
                     idrag := 0, ivent := 0 #)
         ELSIF nav_MM_code = 201 OR nav_MM_code = 202 OR nav_MM_code = 801
            THEN (# deg := 4,
                    ord := 4,
                    DV  := adjusted_DV(DV, GPS_IMU_vel_thresh),
                    idrag := within_vel_thresh(DV, GPS_IMU_vel_thresh),
                    ivent := within_vel_thresh(DV, GPS_IMU_vel_thresh)
                  #)
         ELSIF vector_mag(DV) < GPS_IMU_vel_thresh
            THEN (# deg := 4, ord := 4, DV := zero_vector,
                    idrag := 0, ivent := 0 #)
            ELSE (# deg := 2, ord := 0, DV := DV, idrag := 0, ivent := 0 #)
         ENDIF,

      super_G =
         super_G_GPS(R_GPS_I, V_GPS_I, DV(GM_params), DT2, T_GPS(I),
                     deg(GM_params), ord(GM_params),
                     idrag(GM_params), ivent(GM_params))

    IN (# R_prop := R_prop(super_G),
          V_prop := V_prop(super_G),
          G_prop := G_prop(super_G),
          T_prop := T_current_filt
       #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.3.2:  Propagation of the previous valid receiver states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nav_state_prop_2(I:                  GPS_id,
                 G_two_I:            acceleration_vector,
                 G_two_prev:         GPS_accelerations,
                 GPS_IMU_vel_thresh: real,
                 R_GPS_prev:         GPS_positions,
                 T_current_filt:     mission_time,
                 T_GPS_prev:         GPS_times,
                 V_current_GPS:      GPS_velocities,
                 V_GPS_prev:         GPS_velocities,
                 V_last_GPS_prev:    GPS_velocities) : super_G_prop =

  LET DT  = T_current_filt - T_GPS_prev(I),
      DV  = vector_diff(V_current_GPS(I), V_last_GPS_prev(I)),
      DV2 = adjusted_DV(DV, GPS_IMU_vel_thresh),

      G_ave = scalar_mult(1/2, vector_sum(G_two_I, G_two_prev(I))),
      DV3 = vector_sum(DV2, scalar_mult(DT, G_ave)),
      R_GPS_prev_I =
         vector_sum(R_GPS_prev(I),
                    scalar_mult(DT,
                                vector_sum(V_GPS_prev(I),
                                           scalar_mult(1/2, DV3)))),
      V_GPS_prev_I = vector_sum(V_GPS_prev(I), DV3)

    IN (# R_prop := R_GPS_prev_I,
          V_prop := V_GPS_prev_I,
          G_prop := G_ave,
          T_prop := T_current_filt         %% Placeholder, value not used
       #)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.3.3:  Propagation of the selected GPS state from the previous
%%                  navigation cycle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
nav_state_prop_3(GPS_IMU_vel_thresh:  real,
                 nav_MM_code:         major_mode_code,
                 R_GPS_sel:           position_vector,
                 T_current_filt:      mission_time,
                 T_GPS_sel:           mission_time,
                 V_current_filt:      velocity_vector,
                 V_GPS_sel:           velocity_vector,
                 V_last_GPS_sel:      velocity_vector) : super_G_prop =

  LET DT = T_current_filt - T_GPS_sel,
      DV = vector_diff(V_current_filt, V_last_GPS_sel),

      (GM_params: GM_params) =
         IF nav_MM_code = 201 OR nav_MM_code = 202 OR nav_MM_code = 801
            THEN (# deg   := 4,
                    ord   := 4,
                    DV    := adjusted_DV(DV, GPS_IMU_vel_thresh),
                    idrag := within_vel_thresh(DV, GPS_IMU_vel_thresh),
                    ivent := within_vel_thresh(DV, GPS_IMU_vel_thresh)
                  #)
         ELSIF vector_mag(DV) < GPS_IMU_vel_thresh
            THEN (# deg := 4, ord := 4, DV := zero_vector,
                    idrag := 0, ivent := 0 #)
            ELSE (# deg := 2, ord := 0, DV := DV, idrag := 0, ivent := 0 #)
         ENDIF,

      super_G =
         super_G_GPS(R_GPS_sel, V_GPS_sel, DV(GM_params), DT, T_GPS_sel,
                     deg(GM_params), ord(GM_params),
                     idrag(GM_params), ivent(GM_params))

  IN (# R_prop := R_prop(super_G),
        V_prop := V_prop(super_G),
        G_prop := G_prop(super_G),
        T_prop := T_current_filt
      #)


%%%%%%%%%% Subfunction top-level specification %%%%%%%%%%

GPS_super_G_prop:   TYPE = [GPS_id -> super_G_prop]

nav_state_prop_out: TYPE = [#
      G_two:              GPS_accelerations,
      GPS_DG_SF:          GPS_predicate,
      R_GPS:              GPS_positions,
      R_GPS_prev:         GPS_positions,
      R_GPS_sel:          position_vector,
      T_GPS:              GPS_times,
      T_GPS_sel:          mission_time,
      V_GPS:              GPS_velocities,
      V_GPS_prev:         GPS_velocities,
      V_GPS_sel:          velocity_vector,
      V_last_GPS_sel:     velocity_vector,          %%% state var
      V_last_GPS_two:     velocity_vector           %%% state var
      #]

nav_state_propagation(
       G_two_prev:          GPS_accelerations,     %%% state var
       GPS_DG:              GPS_predicate,
       GPS_DG_SF_prev:      GPS_predicate,         %%% state var
       GPS_IMU_vel_thresh:  real,
       GPS_SV_sel_avail:    bool,                  %%% state var
       GPS_SW_cap:          num_GPS,
       M_WGS84_to_EF:       WGS84_to_EF_matrix,
       nav_MM_code:         major_mode_code,
       R_GPS_ECEF:          GPS_positions_WGS84,
       R_GPS_prev:          GPS_positions,         %%% state var
       R_GPS_sel:           position_vector,       %%% state var
       T_current_filt:      mission_time,
       T_GPS:               GPS_times,
       T_GPS_prev:          GPS_times,             %%% state var
       T_GPS_sel:           mission_time,          %%% state var
       T_last_GPS:          GPS_times,
       V_current_filt:      velocity_vector,
       V_current_GPS:       GPS_velocities,
       V_GPS_ECEF:          GPS_velocities_WGS84,
       V_GPS_prev:          GPS_velocities,        %%% state var
       V_GPS_sel:           velocity_vector,       %%% state var
       V_last_GPS:          GPS_velocities,
       V_last_GPS_prev:     GPS_velocities,
       V_last_GPS_sel:      velocity_vector        %%% state var
       ) : nav_state_prop_out =

  LET (DG_SF: GPS_predicate) = (LAMBDA I: GPS_DG(I) AND (T_GPS(I) >= T_GPS_sel)),

      GPS_prop =
        (LAMBDA I: IF DG_SF(I)
                   THEN nav_state_prop_1(I,
                                         GPS_IMU_vel_thresh,
                                         M_WGS84_to_EF,
                                         nav_MM_code,
                                         R_GPS_ECEF,
                                         T_current_filt,
                                         T_GPS,
                                         T_last_GPS,
                                         V_current_GPS,
                                         V_GPS_ECEF,
                                         V_last_GPS)
```

```
                ELSE null_super_G_prop
              ENDIF),

    prev_prop =
      (LAMBDA I: IF DG_SF(I) AND GPS_DG_SF_prev(I)
                 THEN nav_state_prop_2(I,
                                       G_prop(GPS_prop(I)),
                                       G_two_prev,
                                       GPS_IMU_vel_thresh,
                                       R_GPS_prev,
                                       T_current_filt,
                                       T_GPS_prev,
                                       V_current_GPS,
                                       V_GPS_prev,
                                       V_last_GPS_prev)
                 ELSE null_super_G_prop
              ENDIF),

    sel_prop =
      IF GPS_SV_sel_avail
         THEN nav_state_prop_3(GPS_IMU_vel_thresh,
                               nav_MM_code,
                               R_GPS_sel,
                               T_current_filt,
                               T_GPS_sel,
                               V_current_filt,
                               V_GPS_sel,
                               V_last_GPS_sel)
         ELSE null_super_G_prop
      ENDIF

  IN (# G_two          := (LAMBDA I: G_prop(GPS_prop(I))),
      GPS_DG_SF      := DG_SF,
      R_GPS          := (LAMBDA I: R_prop(GPS_prop(I))),
      R_GPS_prev     := (LAMBDA I: R_prop(prev_prop(I))),
      R_GPS_sel      := R_prop(sel_prop),
      T_GPS          := (LAMBDA I: T_prop(GPS_prop(I))),
      T_GPS_sel      := T_prop(sel_prop),
      V_GPS          := (LAMBDA I: V_prop(GPS_prop(I))),
      V_GPS_prev     := (LAMBDA I: V_prop(prev_prop(I))),
      V_GPS_sel      := V_prop(sel_prop),
      V_last_GPS_sel := V_current_filt,
      V_last_GPS_two := V_current_filt
    #)


END rec_sp_subfuns_1
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rec_sp_subfuns_2.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:21:12 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Wed May 22 09:45:29 1996
%% Update Count    : 154
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%       for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued 3/20/95
%%   Updated to comply with     GPS CR 91051D, issued 6/09/95
%%   Updated to comply with     GPS CR 91051E, issued 6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Receiver State Processing (4.21) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%  Subfunctions, Group 2  %%%%%%%%%%%%%%%%%%%%%%%%%

rec_sp_subfuns_2: THEORY
BEGIN

IMPORTING rec_sp_types


%%%%%%%%%% Common variables %%%%%%%%%%

I:      VAR GPS_id


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.4: GPS State Vector Quality Assessment (3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.5: GPS QA2 Data Computations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

QA2_data_comp_out: TYPE = [#
      DELR_GPS_UVW:        delta_position_vector_UVW,
      DELR_mag:            real,
      DELR_ratio_QA2_max:  real,
      delta_ratio_QA2_ind: display_indicator,
      delta_ratio_QA2_max: real,
      DELV_GPS_UVW:        delta_velocity_vector_UVW,
      DELV_ratio_QA2_max:  real
      #]

null_QA2_data_comp_out:    QA2_data_comp_out

QA2_data_comp(GPS_QA2_RTOL_UVW:  tolerance_vector,
              GPS_QA2_VTOL_UVW:  tolerance_vector,
              M_M50_to_UVW:      M50_to_UVW_matrix,
              R_filt:            position_vector,
              R_in:              position_vector,
              V_filt:            velocity_vector,
              V_in:              velocity_vector) : QA2_data_comp_out =

  LET DELR_I = vector_diff(R_in, R_filt),
      DELV_I = vector_diff(V_in, V_filt),

      (DELR_GPS_UVW: UVW_vector) =
        matrix_vector_mult(M_M50_to_UVW, DELR_I),
      (DELV_GPS_UVW: UVW_vector) =
        matrix_vector_mult(M_M50_to_UVW, DELV_I),

      (DELR_ratio: UVW_vector) =
        (LAMBDA (a: UVW_axis): abs(DELR_GPS_UVW(a)) / GPS_QA2_RTOL_UVW(a)),
      (DELV_ratio: UVW_vector) =
        (LAMBDA (a: UVW_axis): abs(DELV_GPS_UVW(a)) / GPS_QA2_VTOL_UVW(a)),

      DELR_ratio_max = max(DELR_ratio),
      DELV_ratio_max = max(DELV_ratio),

      delta_ratio_max =
         IF DELR_ratio_max > DELV_ratio_max
            THEN DELR_ratio_max
            ELSE DELV_ratio_max
         ENDIF,
      delta_ratio_ind =
         IF DELR_ratio_max > DELV_ratio_max
            THEN P_ind
            ELSE V_ind
         ENDIF

  IN (# DELR_GPS_UVW        := DELR_GPS_UVW,
        DELR_mag            := vector_mag(DELR_I),
        DELR_ratio_QA2_max  := DELR_ratio_max,
        delta_ratio_QA2_ind := delta_ratio_ind,
        delta_ratio_QA2_max := delta_ratio_max,
        DELV_GPS_UVW        := DELV_GPS_UVW,
        DELV_ratio_QA2_max  := DELV_ratio_max
```

```
      #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.1: Second level of GPS receiver state quality assessment, QA2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

state_vect_QA2_out: TYPE = [#
      DELR_GPS_UVW:        delta_position_vector_UVW,
      DELR_mag:            real,
      DELR_ratio_display:  real,
      DELR_ratio_max:      real,
      delta_ratio_ind:     display_indicator,
      delta_ratio_max:     real,
      DELV_GPS_UVW:        delta_velocity_vector_UVW,
      DELV_ratio_display:  real,
      DELV_ratio_max:      real,
      fail_QA2:            bool,
      fail_QA2_R:          bool,
      fail_QA2_V:          bool
      #]

null_state_vect_QA2_out: state_vect_QA2_out

state_vector_QA2(I:                GPS_id,
                 GPS_QA2_RTOL_UVW: tolerance_vector,
                 GPS_QA2_VTOL_UVW: tolerance_vector,
                 M_M50_to_UVW:     M50_to_UVW_matrix,
                 R_filt:           position_vector,
                 R_GPS:            GPS_positions,
                 V_filt:           velocity_vector,
                 V_GPS:            GPS_velocities) : state_vect_QA2_out =

  LET QA2_out = QA2_data_comp(GPS_QA2_RTOL_UVW,
                              GPS_QA2_VTOL_UVW,
                              M_M50_to_UVW,
                              R_filt,
                              R_GPS(I),
                              V_filt,
                              V_GPS(I)),

      fail_QA2_R = DELR_ratio_QA2_max(QA2_out) > 1,
      fail_QA2_V = DELV_ratio_QA2_max(QA2_out) > 1

  IN (# DELR_GPS_UVW       := DELR_GPS_UVW(QA2_out),
        DELR_mag           := DELR_mag(QA2_out),
        DELR_ratio_display := DELR_ratio_QA2_max(QA2_out),
        DELR_ratio_max     := DELR_ratio_QA2_max(QA2_out),
        delta_ratio_ind    := delta_ratio_QA2_ind(QA2_out),
        delta_ratio_max    := delta_ratio_QA2_max(QA2_out),
        DELV_GPS_UVW       := DELV_GPS_UVW(QA2_out),
        DELV_ratio_display := DELV_ratio_QA2_max(QA2_out),
        DELV_ratio_max     := DELV_ratio_QA2_max(QA2_out),
        fail_QA2           := fail_QA2_R OR fail_QA2_V,
        fail_QA2_R         := fail_QA2_R,
        fail_QA2_V         := fail_QA2_V
      #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.2: Third level of GPS receiver state quality assessment, QA3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

state_vect_QA3_out: TYPE = [#
      DELR_ratio_display:  real,
      DELV_ratio_display:  real,
      fail_QA3:            bool,
      fail_QA3_R:          bool,
      fail_QA3_V:          bool
      #]

null_state_vect_QA3_out: state_vect_QA3_out

state_vector_QA3(I:              GPS_id,
                 GPS_QA3_RTOL:   nonzero_real,
                 GPS_QA3_VTOL:   nonzero_real,
                 R_GPS:          GPS_positions,
                 R_GPS_prev:     GPS_positions,
                 V_GPS:          GPS_velocities,
                 V_GPS_prev:     GPS_velocities) : state_vect_QA3_out =

  LET DELR_ratio_display =
        vector_mag(vector_diff(R_GPS(I), R_GPS_prev(I))) / GPS_QA3_RTOL,
      DELV_ratio_display =
        vector_mag(vector_diff(V_GPS(I), V_GPS_prev(I))) / GPS_QA3_VTOL,

      fail_QA3_R = DELR_ratio_display > 1,
      fail_QA3_V = DELV_ratio_display > 1

  IN (# DELR_ratio_display := DELR_ratio_display,
        DELV_ratio_display := DELV_ratio_display,
        fail_QA3           := fail_QA3_R OR fail_QA3_V,
        fail_QA3_R         := fail_QA3_R,
        fail_QA3_V         := fail_QA3_V
      #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.3: GPS State Vector Quality Assessment QA4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
QA4_comp: GPS_predicate =  (LAMBDA I: false)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.4: Computation of QA2 Tolerance Values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

QA2_tolerance_values: TYPE = [#
      DT_QA2:       delta_time,
      max_values:   bool,
      RTOL_UVW:     UVW_vector,
      VTOL_UVW:     UVW_vector
      #]

QA2_tolerance_values(GPS_QA2_RTOL_max_UVW:  tolerance_vector,
                     GPS_QA2_RTOL_UVW_A:    tolerance_vector,
                     GPS_QA2_RTOL_UVW_B:    tolerance_vector,
                     GPS_QA2_RTOL_UVW_C:    tolerance_vector,
                     GPS_QA2_VTOL_max_UVW:  tolerance_vector,
                     GPS_QA2_VTOL_UVW_A:    tolerance_vector,
                     GPS_QA2_VTOL_UVW_B:    tolerance_vector,
                     GPS_QA2_VTOL_UVW_C:    tolerance_vector,
                     max_values:            bool,
                     nav_reset:             bool,
                     T_current_filt:        mission_time,
                     T_GPS_last_update:     mission_time
                     ) : QA2_tolerance_values =

  LET DT_QA2 = T_current_filt - T_GPS_last_update,
      (GPS_QA2_RTOL_UVW: tolerance_vector) =
         vector_sum(GPS_QA2_RTOL_UVW_A,
                    vector_sum(scalar_mult(DT_QA2, GPS_QA2_RTOL_UVW_B),
                               scalar_mult(DT_QA2 * DT_QA2,
                                           GPS_QA2_RTOL_UVW_C))),
      (GPS_QA2_VTOL_UVW: tolerance_vector) =
         vector_sum(GPS_QA2_VTOL_UVW_A,
                    vector_sum(scalar_mult(DT_QA2, GPS_QA2_VTOL_UVW_B),
                               scalar_mult(DT_QA2 * DT_QA2,
                                           GPS_QA2_VTOL_UVW_C))),
      check_max = nav_reset OR NOT max_values,
      max_exceeded =
         (EXISTS (a: UVW_axis):
            GPS_QA2_RTOL_UVW(a) > GPS_QA2_RTOL_max_UVW(a)) OR
         (EXISTS (a: UVW_axis):
            GPS_QA2_VTOL_UVW(a) > GPS_QA2_VTOL_max_UVW(a))

  IN (# DT_QA2     := DT_QA2,
        max_values := max_values OR max_exceeded,
        RTOL_UVW   := IF check_max AND NOT max_exceeded
                      THEN GPS_QA2_RTOL_UVW
                      ELSE GPS_QA2_RTOL_max_UVW
                    ENDIF,
        VTOL_UVW   := IF check_max AND NOT max_exceeded
                      THEN GPS_QA2_VTOL_UVW
                      ELSE GPS_QA2_VTOL_max_UVW
                    ENDIF
     #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.3.3.4.6: GPS GAX Message Computations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GAX_msg_comp_out: TYPE = [#
      QA2_GAX:         GPS_predicate,
      QA3_GAX:         GPS_predicate,
      QA2_msg_cnt:     GPS_counters,
      QA3_msg_cnt:     GPS_counters
      #]

null_GAX_msg_comp_out:    GAX_msg_comp_out

GAX_message_comp(GPS_fail_QA2:          GPS_predicate,
                 GPS_fail_QA3:          GPS_predicate,
                 GPS_GAX_msg_cnt_min:   nat,
                 noop_GPS_GAX_msgs:     bool,
                 QA2_fail_GAX_msg_cnt:  GPS_counters,
                 QA3_fail_GAX_msg_cnt:  GPS_counters) : GAX_msg_comp_out =

  LET QA2_GAX =
        (LAMBDA I: NOT noop_GPS_GAX_msgs AND GPS_fail_QA2(I) AND
                   QA2_fail_GAX_msg_cnt(I) >= GPS_GAX_msg_cnt_min),
      QA3_GAX =
        (LAMBDA I: NOT noop_GPS_GAX_msgs AND GPS_fail_QA3(I) AND
                   QA3_fail_GAX_msg_cnt(I) >= GPS_GAX_msg_cnt_min),

      QA2_msg_cnt =
        (LAMBDA I: IF NOT noop_GPS_GAX_msgs AND GPS_fail_QA2(I)
                      THEN QA2_fail_GAX_msg_cnt(I) + 1
                      ELSE 0
                   ENDIF),
      QA3_msg_cnt =
        (LAMBDA I: IF NOT noop_GPS_GAX_msgs AND GPS_fail_QA3(I)
                      THEN QA3_fail_GAX_msg_cnt(I) + 1
                      ELSE 0
                   ENDIF)

  IN (# QA2_GAX        := QA2_GAX,
        QA3_GAX        := QA3_GAX,
```

```
      QA2_msg_cnt    := QA2_msg_cnt,
      QA3_msg_cnt    := QA3_msg_cnt
    #)


%%%%%%%%%% Subfunction top-level specification %%%%%%%%%%

state_vect_qual_assess_out: TYPE = [#
    DELR_mag:               GPS_ratios,
    DELR_GPS_QA2_UVW:       GPS_delta_pos_UVW,
    DELR_ratio_QA2_display: GPS_ratios,
    DELR_ratio_QA2_max:     GPS_ratios,
    DELR_ratio_QA3_display: GPS_ratios,
    delta_ratio_QA2_ind:    GPS_indicators,
    delta_ratio_QA2_max:    GPS_ratios,
    DELV_GPS_QA2_UVW:       GPS_delta_vel_UVW,
    DELV_ratio_QA2_display: GPS_ratios,
    DELV_ratio_QA2_max:     GPS_ratios,
    DELV_ratio_QA3_display: GPS_ratios,
    DT_QA2:                 delta_time,
    G_two_prev:             GPS_accelerations,   %%% state var
    GPS_DG_SF_prev:         GPS_predicate,       %%% state var
    GPS_fail_QA:            GPS_predicate,
    GPS_fail_QA2:           GPS_predicate,
    GPS_fail_QA2_GAX:       GPS_predicate,
    GPS_fail_QA2_R:         GPS_predicate,
    GPS_fail_QA2_V:         GPS_predicate,
    GPS_fail_QA3:           GPS_predicate,
    GPS_fail_QA3_GAX:       GPS_predicate,
    GPS_fail_QA3_R:         GPS_predicate,
    GPS_fail_QA3_V:         GPS_predicate,
    GPS_QA2_RTOL_UVW:       UVW_vector,          %%% Note 1
    GPS_QA2_VTOL_UVW:       UVW_vector,          %%% Note 1
    M_M50_to_UVW:           M50_to_UVW_matrix,   %%% Note 1
    max_values:             bool,                %%% state var
    QA2_fail_GAX_msg_cnt:   GPS_counters,        %%% state var
    QA3_fail_GAX_msg_cnt:   GPS_counters,        %%% state var
    R_GPS_prev:             GPS_positions,       %%% state var
    T_GPS_prev:             GPS_times,           %%% state var
    V_GPS_prev:             GPS_velocities       %%% state var
    #]

%% Note 1: These outputs are listed to reflect an implicit use of these
%% variables by the State Vector Selection subfunction when it invokes
%% the GPS_QA2_DATA_COMP procedure.  The variables do not appear in the
%% subfunction input-output tables.

state_vector_quality_assessment(
    G_two:                  GPS_accelerations,
    GPS_DG_SF:              GPS_predicate,
    GPS_DG_SF_prev:         GPS_predicate,       %%% state var
    GPS_fail_FOM:           GPS_predicate,
    GPS_GAX_msg_cnt_min:    nat,
    GPS_QA2_RTOL_max_UVW:   tolerance_vector,
    GPS_QA2_RTOL_UVW_A:     tolerance_vector,
    GPS_QA2_RTOL_UVW_B:     tolerance_vector,
    GPS_QA2_RTOL_UVW_C:     tolerance_vector,
    GPS_QA2_VTOL_max_UVW:   tolerance_vector,
    GPS_QA2_VTOL_UVW_A:     tolerance_vector,
    GPS_QA2_VTOL_UVW_B:     tolerance_vector,
    GPS_QA2_VTOL_UVW_C:     tolerance_vector,
    GPS_QA3_RTOL:           nonzero_real,
    GPS_QA3_VTOL:           nonzero_real,
    GPS_SW_cap:             num_GPS,
    max_values:             bool,                %%% state var
    nav_reset:              bool,
    noop_GPS_GAX_msgs:      bool,
    QA2_fail_GAX_msg_cnt:   GPS_counters,        %%% state var
    QA3_fail_GAX_msg_cnt:   GPS_counters,        %%% state var
    R_filt:                 position_vector,
    R_GPS:                  GPS_positions,
    R_GPS_prev:             GPS_positions,
    T_current_filt:         mission_time,
    T_GPS:                  GPS_times,
    T_GPS_last_update:      mission_time,
    V_filt:                 velocity_vector,
    V_GPS:                  GPS_velocities,
    V_GPS_prev:             GPS_velocities
    ) : state_vect_qual_assess_out =

LET M_M50_to_UVW = transpose(UVW_to_M50(R_filt, V_filt)),

    fail_QA4 = QA4_comp,

    tolerances =
       QA2_tolerance_values(GPS_QA2_RTOL_max_UVW,
                            GPS_QA2_RTOL_UVW_A,
                            GPS_QA2_RTOL_UVW_B,
                            GPS_QA2_RTOL_UVW_C,
                            GPS_QA2_VTOL_max_UVW,
                            GPS_QA2_VTOL_UVW_A,
                            GPS_QA2_VTOL_UVW_B,
                            GPS_QA2_VTOL_UVW_C,
                            max_values,
                            nav_reset,
                            T_current_filt,
                            T_GPS_last_update),

    QA2_result =
      (LAMBDA I: IF GPS_DG_SF(I)
```

```
                    THEN state_vector_QA2(I,
                                          RTOL_UVW(tolerances),
                                          VTOL_UVW(tolerances),
                                          M_M50_to_UVW,
                                          R_filt,
                                          R_GPS,
                                          V_filt,
                                          V_GPS)
                    ELSE null_state_vect_QA2_out
                  ENDIF),

    QA3_result =
      (LAMBDA I: IF GPS_DG_SF(I) AND GPS_DG_SF_prev(I)
                    THEN state_vector_QA3(I,
                                          GPS_QA3_RTOL,
                                          GPS_QA3_VTOL,
                                          R_GPS,
                                          R_GPS_prev,
                                          V_GPS,
                                          V_GPS_prev)
                    ELSE null_state_vect_QA3_out
                  ENDIF),

    fail_QA2 = (LAMBDA I: fail_QA2(QA2_result(I))),
    fail_QA3 =
      (LAMBDA I: (NOT GPS_DG_SF_prev(I)) OR fail_QA3(QA3_result(I))),

    fail_QA = (LAMBDA I: GPS_fail_FOM(I) OR fail_QA2(I) OR
                         fail_QA3(I) OR fail_QA4(I)),

    GAX_msgs = GAX_message_comp(fail_QA2,
                                fail_QA3,
                                GPS_GAX_msg_cnt_min,
                                noop_GPS_GAX_msgs,
                                QA2_fail_GAX_msg_cnt,
                                QA3_fail_GAX_msg_cnt)

IN (# DELR_GPS_QA2_UVW       := (LAMBDA I: DELR_GPS_UVW(QA2_result(I))),
      DELR_mag               := (LAMBDA I: DELR_mag(QA2_result(I))),
      DELR_ratio_QA2_display := (LAMBDA I: DELR_ratio_display(QA2_result(I))),
      DELR_ratio_QA2_max     := (LAMBDA I: DELR_ratio_max(QA2_result(I))),
      DELR_ratio_QA3_display := (LAMBDA I: DELR_ratio_display(QA3_result(I))),
      delta_ratio_QA2_ind    := (LAMBDA I: delta_ratio_ind(QA2_result(I))),
      delta_ratio_QA2_max    := (LAMBDA I: delta_ratio_max(QA2_result(I))),
      DELV_GPS_QA2_UVW       := (LAMBDA I: DELV_GPS_UVW(QA2_result(I))),
      DELV_ratio_QA2_display := (LAMBDA I: DELV_ratio_display(QA2_result(I))),
      DELV_ratio_QA2_max     := (LAMBDA I: DELV_ratio_max(QA2_result(I))),
      DELV_ratio_QA3_display := (LAMBDA I: DELV_ratio_display(QA3_result(I))),
      DT_QA2                 := DT_QA2(tolerances),
      G_two_prev             :=
       (LAMBDA I: IF GPS_DG_SF(I) THEN G_two(I) ELSE null_acceleration ENDIF),
      GPS_DG_SF_prev         := GPS_DG_SF,
      GPS_fail_QA            := fail_QA,
      GPS_fail_QA2           := fail_QA2,
      GPS_fail_QA2_GAX       := QA2_GAX(GAX_msgs),
      GPS_fail_QA2_R         := (LAMBDA I: fail_QA2_R(QA2_result(I))),
      GPS_fail_QA2_V         := (LAMBDA I: fail_QA2_V(QA2_result(I))),
      GPS_fail_QA3           := fail_QA3,
      GPS_fail_QA3_GAX       := QA3_GAX(GAX_msgs),
      GPS_fail_QA3_R         := (LAMBDA I: fail_QA3_R(QA3_result(I))),
      GPS_fail_QA3_V         := (LAMBDA I: fail_QA3_V(QA3_result(I))),
      GPS_QA2_RTOL_UVW       := RTOL_UVW(tolerances),
      GPS_QA2_VTOL_UVW       := VTOL_UVW(tolerances),
      M_M50_to_UVW           := M_M50_to_UVW,
      max_values             := max_values(tolerances),
      QA2_fail_GAX_msg_cnt   := QA2_msg_cnt(GAX_msgs),
      QA3_fail_GAX_msg_cnt   := QA3_msg_cnt(GAX_msgs),
      R_GPS_prev             :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I) ELSE null_position ENDIF),
      T_GPS_prev             :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN T_GPS(I) ELSE null_mission_time ENDIF),
      V_GPS_prev             :=
        (LAMBDA I: IF GPS_DG_SF(I) THEN V_GPS(I) ELSE null_velocity ENDIF)
    #)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.5: GPS State Vector Selection (4)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SV_sel_step_1_out: TYPE = [#
      DELR_SV_sel:           real,
      delta_ratio_ind_sel:   display_indicator,
      delta_ratio_SV_sel:    real,
      FOM_SV_sel:            GPS_figure_of_merit,
      GPS_sel_DELR_UVW:      delta_position_vector_UVW,
      GPS_sel_DELV_UVW:      delta_velocity_vector_UVW,
      GPS_SV_sel_avail:      bool,
      MVS_rcvr_sel:          nat,
      R_GPS_sel:             position_vector,
      sel_rcvr:              GPS_predicate,
      T_GPS_off:             mission_time,
      T_GPS_sel:             mission_time,
      V_GPS_sel:             velocity_vector
      #]

%%%%%%%%%% Steps 3.0, 4.0 %%%%%%%%%%

state_vector_sel_step_1(DELR_GPS_QA2_UVW:     GPS_delta_pos_UVW,
```

```
                      DELR_mag:             GPS_ratios,
                      delta_ratio_QA2_ind:  GPS_indicators,
                      delta_ratio_QA2_max:  GPS_ratios,
                      DELV_GPS_QA2_UVW:     GPS_delta_vel_UVW,
                      FOM:                  GPS_FOM_vector,
                      GPS_SV_sel_avail:     bool,
                      nautmi_per_ft:        real,
                      R_GPS:                GPS_positions,
                      T_current_filt:       mission_time,
                      T_GPS:                GPS_times,
                      V_GPS:                GPS_velocities
                      ) : SV_sel_step_1_out =

  (# DELR_SV_sel         := DELR_mag(2) * nautmi_per_ft,
     delta_ratio_ind_sel := delta_ratio_QA2_ind(2),
     delta_ratio_SV_sel  := delta_ratio_QA2_max(2),
     FOM_SV_sel          := FOM(2),
     GPS_sel_DELR_UVW    := DELR_GPS_QA2_UVW(2),
     GPS_sel_DELV_UVW    := DELV_GPS_QA2_UVW(2),
     GPS_SV_sel_avail    := true,
     MVS_rcvr_sel        := 2,
     R_GPS_sel           := R_GPS(2),
     sel_rcvr            := (LAMBDA I: I = 2),
     T_GPS_off           := T_current_filt,
     T_GPS_sel           := IF GPS_SV_sel_avail
                            THEN T_GPS(2)
                            ELSE T_current_filt
                          ENDIF,
     V_GPS_sel           := V_GPS(2)
  #)

%%%%%%%%%% Steps 3.1, 4.1 %%%%%%%%%%

state_vector_sel_step_2(DELR_mag:             GPS_ratios,
                      delta_ratio_QA2_ind:  GPS_indicators,
                      delta_ratio_QA2_max:  GPS_ratios,
                      GPS_mode_rcvd:        GPS_rcvr_mode_vector,
                      GPS_QA2_RTOL_UVW:     tolerance_vector,
                      GPS_QA2_VTOL_UVW:     tolerance_vector,
                      GPS_SV_sel_avail:     bool,
                      M_M50_to_UVW:         M50_to_UVW_matrix,
                      nautmi_per_ft:        real,
                      R_filt:               position_vector,
                      R_GPS_sel:            position_vector,
                      T_current_filt:       mission_time,
                      T_GPS_off:            mission_time,
                      T_GPS_sel:            mission_time,
                      V_filt:               velocity_vector,
                      V_GPS_sel:            velocity_vector
                      ) : SV_sel_step_1_out =

  LET QA2_out = IF GPS_SV_sel_avail
                THEN QA2_data_comp(GPS_QA2_RTOL_UVW,
                                   GPS_QA2_VTOL_UVW,
                                   M_M50_to_UVW,
                                   R_filt,
                                   R_GPS_sel,
                                   V_filt,
                                   V_GPS_sel)
                ELSE null_QA2_data_comp_out
              ENDIF

  IN (# DELR_SV_sel         := DELR_mag(QA2_out) * nautmi_per_ft,
        delta_ratio_ind_sel := delta_ratio_QA2_ind(QA2_out),
        delta_ratio_SV_sel  := delta_ratio_QA2_max(QA2_out),
        FOM_SV_sel          := 1,                  %% Value unspecified
        GPS_sel_DELR_UVW    := DELR_GPS_UVW(QA2_out),
        GPS_sel_DELV_UVW    := DELV_GPS_UVW(QA2_out),
        GPS_SV_sel_avail    := GPS_SV_sel_avail,
        MVS_rcvr_sel        := 0,
        R_GPS_sel           := R_GPS_sel,
        sel_rcvr            := (LAMBDA I: false),
        T_GPS_off           := IF GPS_mode_rcvd(2) /= nav AND
                                   NOT GPS_SV_sel_avail
                                   THEN T_current_filt
                                   ELSE T_GPS_off
                                ENDIF,
        T_GPS_sel           := T_GPS_sel,
        V_GPS_sel           := V_GPS_sel
     #)

%%%%%%%%%% Steps 7.1 to 7.4 %%%%%%%%%%

SV_sel_step_3_out: TYPE = [#
     FOM_nav:             GPS_FOM_vector,
     GPS_fail_FOM_disp:   GPS_predicate,
     GPS_lat_ind:         display_indicator,
     GPS_lon_ind:         display_indicator,
     GPS_sel_alt:         distance,
     GPS_sel_alt_disp:    distance,
     GPS_sel_lat:         latitude,
     GPS_sel_lat_disp:    latitude,
     GPS_sel_lon:         longitude,
     GPS_sel_lon_disp:    longitude,
     SV_sel_stat:         display_status
     #]

null_SV_sel_step_3_out:   SV_sel_step_3_out

state_vector_sel_step_3(deg_to_rad:         real,
```

```
                        delta_ratio_SV_sel:  real,
                        FOM:                 GPS_FOM_vector,
                        GPS_fail_FOM:        GPS_predicate,
                        nautmi_per_ft:       real,
                        QA_override:         GPS_predicate,
                        R_GPS_sel:           position_vector,
                        sel_cmd:             GPS_predicate,
                        T_GPS_sel:           mission_time
                        ) : SV_sel_step_3_out =

  LET sel_stat = IF (EXISTS I: sel_cmd(I)) AND QA_override(2)
                    THEN up_arrow_stat
                 ELSIF delta_ratio_SV_sel < 1
                    THEN blank_stat
                    ELSE down_arrow_stat
                 ENDIF,

      geodetic_params =
        EF_to_geodetic(matrix_vector_mult(
                        transpose(earth_fixed_to_M50_coord(T_GPS_sel)),
                        R_GPS_sel)),

      lat_ind = IF lat(geodetic_params) >= 0 THEN N_ind ELSE S_ind ENDIF,
      lon_ind = IF lon(geodetic_params) >= 0 THEN E_ind ELSE W_ind ENDIF

  IN (# FOM_nav              := FOM,
        GPS_fail_FOM_disp    := GPS_fail_FOM,
        GPS_lat_ind          := lat_ind,
        GPS_lon_ind          := lon_ind,
        GPS_sel_alt          := alt(geodetic_params),
        GPS_sel_alt_disp     := alt(geodetic_params) * nautmi_per_ft,
        GPS_sel_lat          := lat(geodetic_params),
        GPS_sel_lat_disp     := abs(lat(geodetic_params)) * deg_to_rad,
        GPS_sel_lon          := lon(geodetic_params),
        GPS_sel_lon_disp     := abs(lon(geodetic_params)) * deg_to_rad,
        SV_sel_stat          := sel_stat
      #)

%%%%%%%%%% Step 7.5 %%%%%%%%%%

SV_sel_step_4_out: TYPE = [#
     corr_coeff_GPS:      corr_coeff_vector,
     sig_diag_GPS:        cov_diagonal_vector
     #]

null_SV_sel_step_4_out:    SV_sel_step_4_out

state_vector_sel_step_4(corr_coeff_GPS_nom:  corr_coeff_vector,
                        FOM_cov_SF_table:    FOM_cov_SF_table,
                        FOM_SV_sel:          GPS_figure_of_merit,
                        sig_diag_GPS_nom:    cov_diagonal_vector
                        ) : SV_sel_step_4_out =

  LET FOM_pos_SF = FOM_cov_SF_table(1)(FOM_SV_sel),
      FOM_vel_SF = FOM_cov_SF_table(2)(FOM_SV_sel),

      (sig_diag_GPS: cov_diagonal_vector) =
        (LAMBDA (k: cov_diagonal_index):
          IF k <= 3
             THEN FOM_pos_SF * sig_diag_GPS_nom(k)
             ELSE FOM_vel_SF * sig_diag_GPS_nom(k)
          ENDIF)

  IN (# corr_coeff_GPS      := corr_coeff_GPS_nom,
        sig_diag_GPS        := sig_diag_GPS
      #)


%%%%%%%%%% Subfunction top-level specification %%%%%%%%%%

state_vect_sel_out: TYPE = [#
     corr_coeff_GPS:          corr_coeff_vector,
     crew_des_rcvr_rcvd:      GPS_predicate,
     crew_QA_override_rcvd:   GPS_predicate,
     crew_QA_ovrd_status_ind: GPS_predicate,
     DELR_SV_sel:             real,
     delta_ratio_SV_sel:      real,
     delta_ratio_SV_sel_ind:  display_indicator,
     delta_ratio_SV_sel_stat: display_status,
     DT_GPS_off_min:          delta_time,
     FOM_nav:                 GPS_FOM_vector,
     FOM_SV_sel:              GPS_figure_of_merit,
     GPS_fail_FOM_disp:       GPS_predicate,
     GPS_off_alert:           bool,
     GPS_off_alert_GAX:       bool,
     GPS_sel_alt:             distance,
     GPS_sel_alt_disp:        distance,
     GPS_sel_DELR_UVW:        delta_position_vector_UVW,
     GPS_sel_DELV_UVW:        delta_velocity_vector_UVW,
     GPS_sel_lat:             latitude,
     GPS_sel_lat_disp:        latitude,
     GPS_sel_lat_ind:         display_indicator,
     GPS_sel_lon:             longitude,
     GPS_sel_lon_disp:        longitude,
     GPS_sel_lon_ind:         display_indicator,
     GPS_SV_sel_avail:        bool,                    %%% Note 2
     MVS_rcvr_sel:            nat,
     num_GPS_sel:             nat,
     off_alert_GAX_msg_cnt:   nat,                     %%% state var
     R_GPS_sel:               position_vector,         %%% state var
```

```
      sel_cmd:                GPS_predicate,
      sel_rcvr:               GPS_predicate,
      sig_diag_GPS:           cov_diagonal_vector,
      T_GPS_off:              mission_time,              %%% state var
      T_GPS_sel:              mission_time,              %%% state var
      V_GPS_sel:              velocity_vector            %%% state var
      #]

%% Note 2: This output is both a state variable and a principal
%% function output.

%% Note 3: These inputs are listed to reflect an implicit use of these
%% variables by the State Vector Selection subfunction when it invokes
%% the GPS_QA2_DATA_COMP procedure.  The variables do not appear in the
%% subfunction input-output tables.

state_vector_selection(
        corr_coeff_GPS_nom:     corr_coeff_vector,
        crew_deselect_rcvr:     GPS_predicate,
        deg_to_rad:             real,
        DELR_GPS_QA2_UVW:       GPS_delta_pos_UVW,
        DELR_mag:               GPS_ratios,
        delta_ratio_QA2_ind:    GPS_indicators,
        delta_ratio_QA2_max:    GPS_ratios,
        DELV_GPS_QA2_UVW:       GPS_delta_vel_UVW,
        FOM:                    GPS_FOM_vector,
        FOM_cov_SF_table:       FOM_cov_SF_table,
        GPS_AIF_rcvd:           AIF_flag,
        GPS_alert_DT:           delta_time,
        GPS_DG_SF:              GPS_predicate,
        GPS_fail_FOM:           GPS_predicate,
        GPS_fail_QA:            GPS_predicate,
        GPS_GAX_msg_cnt_min:    nat,
        GPS_mode_rcvd:          GPS_rcvr_mode_vector,
        GPS_QA2_RTOL_UVW:       tolerance_vector,      %%% Note 3
        GPS_QA2_VTOL_UVW:       tolerance_vector,      %%% Note 3
        GPS_SV_sel_avail:       bool,                  %%% state var
        GPS_SW_cap:             num_GPS,
        M_M50_to_UVW:           M50_to_UVW_matrix,     %%% Note 3
        nautmi_per_ft:          real,
        noop_GPS_GAX_msgs:      bool,
        off_alert_GAX_msg_cnt:  nat,                   %%% state var
        QA_override:            GPS_predicate,
        R_filt:                 position_vector,       %%% Note 3
        R_GPS:                  GPS_positions,
        R_GPS_sel:              position_vector,
        sig_diag_GPS_nom:       cov_diagonal_vector,
        T_current_filt:         mission_time,
        T_GPS:                  GPS_times,
        T_GPS_off:              mission_time,          %%% state var
        T_GPS_sel:              mission_time,
        V_filt:                 velocity_vector,       %%% Note 3
        V_GPS:                  GPS_velocities,
        V_GPS_sel:              velocity_vector
        ) : state_vect_sel_out =

LET sel_cmd =
      (LAMBDA I: GPS_DG_SF(I) AND NOT crew_deselect_rcvr(I) AND
                 (GPS_AIF_rcvd = force OR QA_override(I) OR
                  NOT GPS_fail_QA(I))),
    ovrd_status_ind =
      (LAMBDA I: (NOT GPS_DG_SF(I) OR crew_deselect_rcvr(I)) AND
                 QA_override(I)),

    sel_step_1 =
       IF (EXISTS I: sel_cmd(I))
          THEN state_vector_sel_step_1(DELR_GPS_QA2_UVW,
                                       DELR_mag,
                                       delta_ratio_QA2_ind,
                                       delta_ratio_QA2_max,
                                       DELV_GPS_QA2_UVW,
                                       FOM,
                                       GPS_SV_sel_avail,
                                       nautmi_per_ft,
                                       R_GPS,
                                       T_current_filt,
                                       T_GPS,
                                       V_GPS)
          ELSE state_vector_sel_step_2(DELR_mag,
                                       delta_ratio_QA2_ind,
                                       delta_ratio_QA2_max,
                                       GPS_mode_rcvd,
                                       GPS_QA2_RTOL_UVW,
                                       GPS_QA2_VTOL_UVW,
                                       GPS_SV_sel_avail,
                                       M_M50_to_UVW,
                                       nautmi_per_ft,
                                       R_filt,
                                       R_GPS_sel,
                                       T_current_filt,
                                       T_GPS_off,
                                       T_GPS_sel,
                                       V_filt,
                                       V_GPS_sel)
       ENDIF,

    DT_GPS_off = T_current_filt - T_GPS_off(sel_step_1),
    DT_GPS_off_min = DT_GPS_off / 60,

    GPS_off_alert = (DT_GPS_off >= GPS_alert_DT),
```

```
    GPS_off_alert_GAX =
      (DT_GPS_off >= GPS_alert_DT AND
       NOT noop_GPS_GAX_msgs AND
       off_alert_GAX_msg_cnt >= GPS_GAX_msg_cnt_min),

    off_alert_GAX_cnt =
      IF DT_GPS_off < GPS_alert_DT OR noop_GPS_GAX_msgs
         THEN 0
      ELSIF off_alert_GAX_msg_cnt >= GPS_GAX_msg_cnt_min
         THEN off_alert_GAX_msg_cnt
         ELSE off_alert_GAX_msg_cnt + 1
      ENDIF,

    sel_step_3 =
       IF GPS_SV_sel_avail
          THEN state_vector_sel_step_3(deg_to_rad,
                                       delta_ratio_SV_sel(sel_step_1),
                                       FOM,
                                       GPS_fail_FOM,
                                       nautmi_per_ft,
                                       QA_override,
                                       R_GPS_sel(sel_step_1),
                                       sel_cmd,
                                       T_GPS_sel(sel_step_1))
          ELSE null_SV_sel_step_3_out
       ENDIF,

    sel_step_4 =
       IF (EXISTS I: sel_cmd(I))
          THEN state_vector_sel_step_4(corr_coeff_GPS_nom,
                                       FOM_cov_SF_table,
                                       FOM_SV_sel(sel_step_1),
                                       sig_diag_GPS_nom)
          ELSE null_SV_sel_step_4_out
       ENDIF

IN (# corr_coeff_GPS          := corr_coeff_GPS(sel_step_4),
      crew_des_rcvr_rcvd      := crew_deselect_rcvr,
      crew_QA_override_rcvd   := QA_override,
      crew_QA_ovrd_status_ind := ovrd_status_ind,
      DELR_SV_sel             := DELR_SV_sel(sel_step_1),
      delta_ratio_SV_sel      := delta_ratio_SV_sel(sel_step_1),
      delta_ratio_SV_sel_ind  := delta_ratio_ind_sel(sel_step_1),
      delta_ratio_SV_sel_stat := SV_sel_stat(sel_step_3),
      DT_GPS_off_min          := DT_GPS_off_min,
      FOM_nav                 := FOM_nav(sel_step_3),
      FOM_SV_sel              := FOM_SV_sel(sel_step_1),
      GPS_fail_FOM_disp       := GPS_fail_FOM_disp(sel_step_3),
      GPS_off_alert           := GPS_off_alert,
      GPS_off_alert_GAX       := GPS_off_alert_GAX,
      GPS_sel_alt             := GPS_sel_alt(sel_step_3),
      GPS_sel_alt_disp        := GPS_sel_alt_disp(sel_step_3),
      GPS_sel_DELR_UVW        := GPS_sel_DELR_UVW(sel_step_1),
      GPS_sel_DELV_UVW        := GPS_sel_DELV_UVW(sel_step_1),
      GPS_sel_lat             := GPS_sel_lat(sel_step_3),
      GPS_sel_lat_disp        := GPS_sel_lat_disp(sel_step_3),
      GPS_sel_lat_ind         := GPS_lat_ind(sel_step_3),
      GPS_sel_lon             := GPS_sel_lon(sel_step_3),
      GPS_sel_lon_disp        := GPS_sel_lon_disp(sel_step_3),
      GPS_sel_lon_ind         := GPS_lon_ind(sel_step_3),
      GPS_SV_sel_avail        := GPS_SV_sel_avail(sel_step_1),
      MVS_rcvr_sel            := MVS_rcvr_sel(sel_step_1),
      num_GPS_sel             :=
         IF (EXISTS I: sel_cmd(I)) THEN 1 ELSE 0 ENDIF,
      off_alert_GAX_msg_cnt   := off_alert_GAX_cnt,
      R_GPS_sel               := R_GPS_sel(sel_step_1),
      sel_cmd                 := sel_cmd,
      sel_rcvr                := sel_rcvr(sel_step_1),
      sig_diag_GPS            := sig_diag_GPS(sel_step_4),
      T_GPS_off               := T_GPS_off(sel_step_1),
      T_GPS_sel               := T_GPS_sel(sel_step_1),
      V_GPS_sel               := V_GPS_sel(sel_step_1)
    #)


END rec_sp_subfuns_2
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rec_sp_subfuns_3.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:21:12 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Wed May 22 09:45:11 1996
%% Update Count    : 147
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%       for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued  3/20/95
%%   Updated to comply with     GPS CR 91051D, issued  6/09/95
%%   Updated to comply with     GPS CR 91051E, issued  6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Receiver State Processing (4.21) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%  Subfunctions, Group 3  %%%%%%%%%%%%%%%%%%%%%%%%

rec_sp_subfuns_3: THEORY
BEGIN

IMPORTING rec_sp_types


%%%%%%%%%% Common variables %%%%%%%%%%

I:      VAR GPS_id


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.6: GPS Reference State Announced Reset (5)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ref_state_anncd_reset_out: TYPE = [#
      G_ref_anncd_reset:        GPS_accelerations,
      GPS_anncd_reset:          GPS_predicate,
      GPS_anncd_reset_avail:    GPS_predicate,
      R_ref_anncd_reset:        GPS_positions,
      T_anncd_reset:            GPS_times,                    %%% state var
      T_ref_anncd_reset:        GPS_times,
      V_IMU_ref_anncd_reset:    GPS_velocities,
      V_ref_anncd_reset:        GPS_velocities
      #]

ref_state_announced_reset(DT_anncd_reset:  delta_time,
                          G_two:           GPS_accelerations,
                          GPS_DG_SF:       GPS_predicate,
                          GPS_SW_cap:      num_GPS,
                          R_GPS:           GPS_positions,
                          T_anncd_reset:   GPS_times,          %%% state var
                          T_current_filt:  mission_time,
                          T_GPS:           GPS_times,
                          V_current_GPS:   GPS_velocities,
                          V_GPS:           GPS_velocities
                          ) : ref_state_anncd_reset_out =

  (# G_ref_anncd_reset     :=
       (LAMBDA I: IF GPS_DG_SF(I) THEN G_two(I) ELSE null_acceleration ENDIF),
     GPS_anncd_reset       :=
       (LAMBDA I: GPS_DG_SF(I) AND
                  (T_current_filt - T_anncd_reset(I) > DT_anncd_reset)),
     GPS_anncd_reset_avail := GPS_DG_SF,
     R_ref_anncd_reset     :=
       (LAMBDA I: IF GPS_DG_SF(I) THEN R_GPS(I) ELSE null_position ENDIF),
     T_anncd_reset         :=
       (LAMBDA I: IF GPS_DG_SF(I) AND
                     (T_current_filt - T_anncd_reset(I) > DT_anncd_reset)
                     THEN T_current_filt
                     ELSE null_mission_time
                  ENDIF),
     T_ref_anncd_reset     :=
       (LAMBDA I: IF GPS_DG_SF(I) THEN T_GPS(I) ELSE null_mission_time ENDIF),
     V_IMU_ref_anncd_reset :=
       (LAMBDA I: IF GPS_DG_SF(I)
                     THEN V_current_GPS(I)
                     ELSE null_velocity
                  ENDIF),
     V_ref_anncd_reset     :=
       (LAMBDA I: IF GPS_DG_SF(I) THEN V_GPS(I) ELSE null_velocity ENDIF)
   #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.3.3.7: GPS Downlist Computation (6)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pack_collection_word(crew_deselect_rcvr:    GPS_predicate,
                     GPS_AIF_rcvd:          AIF_flag,
                     GPS_DG_SF:             GPS_predicate,
                     GPS_fail_FOM:          GPS_predicate,
                     GPS_off_alert:         bool,
                     GPS_to_nav_force_rcvd: GPS_predicate,
```

```
                    GPS_to_nav_rcvd:      GPS_predicate,
                    MVS_rcvr_sel:         nat) : collection_word

pack_collection_word_two(FOM_SV_sel:       GPS_figure_of_merit,
                         GPS_fail_QA:      GPS_predicate,
                         GPS_fail_QA2:     GPS_predicate,
                         GPS_fail_QA2_R:   GPS_predicate,
                         GPS_fail_QA2_V:   GPS_predicate,
                         GPS_fail_QA3:     GPS_predicate,
                         GPS_fail_QA3_R:   GPS_predicate,
                         GPS_fail_QA3_V:   GPS_predicate
                         ) : collection_word


GPS_downlist_out: TYPE = [#
      DELR_ratio_QA2_DL_2:     int,              %%% Note 1
      DELR_ratio_QA3_DL_2:     int,              %%% Note 1
      DELV_ratio_QA2_DL_2:     int,              %%% Note 1
      DELV_ratio_QA3_DL_2:     int,              %%% Note 1
      DT_QA2_DL:               int,
      GPS_collection_word:     collection_word,
      GPS_collection_word_two: collection_word,
      GPS_QA2_RTOL_UVW_DL:     UVW_int_vector,
      GPS_QA2_VTOL_UVW_DL:     UVW_int_vector,
      GPS_sel_alt_DL:          int,
      GPS_sel_DELR_UVW_DL:     UVW_int_vector,
      GPS_sel_DELV_UVW_DL:     UVW_int_vector,
      GPS_sel_lat_DL:          int,
      GPS_sel_lon_DL:          int,
      R_GPS_sel_DL:            M50_int_vector,
      T_GPS_sel_DL:            real,
      V_GPS_sel_DL:            M50_int_vector
      #]

%% Note 1: Only element 2 of these vectors are provided as outputs.

GPS_downlist_computation(alt_SF:                nonzero_real,
                         ang_SF:                nonzero_real,
                         crew_deselect_rcvr:    GPS_predicate,
                         DELR_ratio_QA2_max:    GPS_ratios,
                         DELR_ratio_QA3_display: GPS_ratios,
                         DELR_UVW_pos_SF:       nonzero_real,
                         DELV_ratio_QA2_max:    GPS_ratios,
                         DELV_ratio_QA3_display: GPS_ratios,
                         DELV_UVW_vel_SF:       nonzero_real,
                         DT_QA2:                delta_time,
                         DT_QA2_SF:             nonzero_real,
                         FOM_SV_sel:            GPS_figure_of_merit,
                         GPS_AIF_rcvd:          AIF_flag,
                         GPS_DG_SF:             GPS_predicate,
                         GPS_fail_FOM:          GPS_predicate,
                         GPS_fail_QA:           GPS_predicate,
                         GPS_fail_QA2:          GPS_predicate,
                         GPS_fail_QA2_R:        GPS_predicate,
                         GPS_fail_QA2_V:        GPS_predicate,
                         GPS_fail_QA3:          GPS_predicate,
                         GPS_fail_QA3_R:        GPS_predicate,
                         GPS_fail_QA3_V:        GPS_predicate,
                         GPS_off_alert:         bool,
                         GPS_QA2_RTOL_UVW:      tolerance_vector,
                         GPS_QA2_VTOL_UVW:      tolerance_vector,
                         GPS_sel_alt:           distance,
                         GPS_sel_DELR_UVW:      delta_position_vector_UVW,
                         GPS_sel_DELV_UVW:      delta_velocity_vector_UVW,
                         GPS_sel_lat:           latitude,
                         GPS_sel_lon:           longitude,
                         GPS_to_nav_rcvd:       GPS_predicate,
                         GPS_to_nav_force_rcvd: GPS_predicate,
                         MVS_rcvr_sel:          nat,
                         nautmi_per_ft:         real,
                         QA2_DELR_ratio_SF:     nonzero_real,
                         QA2_DELV_ratio_SF:     nonzero_real,
                         QA2_RTOL_SF:           nonzero_real,
                         QA2_VTOL_SF:           nonzero_real,
                         QA3_DELR_ratio_SF:     nonzero_real,
                         QA3_DELV_ratio_SF:     nonzero_real,
                         R_GPS_sel:             position_vector,
                         SF_pos:                nonzero_real,
                         SF_vel:                nonzero_real,
                         T_GPS_sel:             mission_time,
                         V_GPS_sel:             velocity_vector
                         ) : GPS_downlist_out =

(# DELR_ratio_QA2_DL_2 :=
     truncate(DELR_ratio_QA2_max(2) / QA2_DELR_ratio_SF),
   DELR_ratio_QA3_DL_2 :=
     truncate(DELR_ratio_QA3_display(2) / QA3_DELR_ratio_SF),
   DELV_ratio_QA2_DL_2 :=
     truncate(DELV_ratio_QA2_max(2) / QA2_DELV_ratio_SF),
   DELV_ratio_QA3_DL_2 :=
     truncate(DELV_ratio_QA3_display(2) / QA3_DELV_ratio_SF),
   DT_QA2_DL           := truncate(DT_QA2 / DT_QA2_SF),
   GPS_collection_word :=
      pack_collection_word(crew_deselect_rcvr,
                           GPS_AIF_rcvd,
                           GPS_DG_SF,
                           GPS_fail_FOM,
                           GPS_off_alert,
                           GPS_to_nav_force_rcvd,
                           GPS_to_nav_rcvd,
```

```
                              MVS_rcvr_sel),
      GPS_collection_word_two :=
         pack_collection_word_two(FOM_SV_sel,
                                  GPS_fail_QA,
                                  GPS_fail_QA2,
                                  GPS_fail_QA2_R,
                                  GPS_fail_QA2_V,
                                  GPS_fail_QA3,
                                  GPS_fail_QA3_R,
                                  GPS_fail_QA3_V),
      GPS_QA2_RTOL_UVW_DL :=
         (LAMBDA (a: UVW_axis): truncate(GPS_QA2_RTOL_UVW(a) / QA2_RTOL_SF)),
      GPS_QA2_VTOL_UVW_DL :=
         (LAMBDA (a: UVW_axis): truncate(GPS_QA2_VTOL_UVW(a) / QA2_VTOL_SF)),
      GPS_sel_alt_DL      := truncate(GPS_sel_alt * nautmi_per_ft / alt_SF),
      GPS_sel_DELR_UVW_DL :=
         (LAMBDA (a: UVW_axis): truncate(GPS_sel_DELR_UVW(a) / DELR_UVW_pos_SF)),
      GPS_sel_DELV_UVW_DL :=
         (LAMBDA (a: UVW_axis): truncate(GPS_sel_DELV_UVW(a) / DELV_UVW_vel_SF)),
      GPS_sel_lat_DL      := truncate(GPS_sel_lat / ang_SF),
      GPS_sel_lon_DL      := truncate(GPS_sel_lon / ang_SF),
      R_GPS_sel_DL        :=
         (LAMBDA (a: M50_axis): truncate(R_GPS_sel(a) / SF_pos)),
      T_GPS_sel_DL        := T_GPS_sel,
      V_GPS_sel_DL        :=
         (LAMBDA (a: M50_axis): truncate(V_GPS_sel(a) / SF_vel))
   #)


END rec_sp_subfuns_3
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rec_sp_pf.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:21:12 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Wed May 22 09:44:43 1996
%% Update Count    : 145
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%      for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued 3/20/95
%%   Updated to comply with     GPS CR 91051D, issued 6/09/95
%%   Updated to comply with     GPS CR 91051E, issued 6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Receiver State Processing (4.21) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%  Principal Function  %%%%%%%%%%%%%%%%%%%%%%%%%

rec_sp_pf: THEORY
BEGIN

IMPORTING rec_sp_types,
          rec_sp_subfuns_1, rec_sp_subfuns_2, rec_sp_subfuns_3

rec_sp_inputs: TYPE = [#
     crew_deselect_rcvr:     GPS_predicate,
     FOM:                    GPS_FOM_vector,
     GPS_AIF_rcvd:           AIF_flag,
     GPS_DG:                 GPS_predicate,
     GPS_fail_FOM:           GPS_predicate,
     GPS_mode_rcvd:          GPS_rcvr_mode_vector,
     GPS_SV_sel_avail:       bool,
     GPS_to_nav_force_rcvd:  GPS_predicate,
     GPS_to_nav_rcvd:        GPS_predicate,
     nav_IMU_to_GPS:         GPS_IMU_ids,
     nav_MM_code:            major_mode_code,
     nav_reset:              bool,
     noop_GPS_GAX_msgs:      bool,
     QA_override:            GPS_predicate,
     R_filt:                 position_vector,
     R_GPS_ECEF:             GPS_positions_WGS84,
     T_current_filt:         mission_time,
     T_GPS:                  GPS_times,
     T_GPS_last_update:      mission_time,
     T_last_GPS:             GPS_times,
     V_current_filt:         velocity_vector,
     V_filt:                 velocity_vector,
     V_GPS_ECEF:             GPS_velocities_WGS84,
     V_last_GPS:             GPS_velocities
     #]

rec_sp_state: TYPE = [#
     G_two_prev:             GPS_accelerations,
     GPS_DG_SF_prev:         GPS_predicate,
     GPS_SV_sel_avail:       bool,
     max_values:             bool,
     off_alert_GAX_msg_cnt:  nat,
     QA2_fail_GAX_msg_cnt:   GPS_counters,
     QA3_fail_GAX_msg_cnt:   GPS_counters,
     R_GPS_prev:             GPS_positions,
     R_GPS_sel:              position_vector,
     T_anncd_reset:          GPS_times,
     T_GPS_off:              mission_time,
     T_GPS_prev:             GPS_times,
     T_GPS_sel:              mission_time,
     V_GPS_prev:             GPS_velocities,
     V_GPS_sel:              velocity_vector,
     V_last_GPS_sel:         velocity_vector,
     V_last_GPS_two:         velocity_vector
     #]

rec_sp_I_loads: TYPE = [#
     alt_SF:                 nonzero_real,
     ang_SF:                 nonzero_real,
     corr_coeff_GPS_nom:     corr_coeff_vector,
     DELR_UVW_pos_SF:        nonzero_real,
     DELV_UVW_vel_SF:        nonzero_real,
     DT_QA2_SF:              nonzero_real,
     DT_anncd_reset:         delta_time,
     FOM_cov_SF_table:       FOM_cov_SF_table,
     GPS_alert_DT:           delta_time,
     GPS_GAX_msg_cnt_min:    nat,
     GPS_IMU_vel_thresh:     real,
     GPS_installed:          GPS_predicate,  %% elements 1,3 are K-loads
     GPS_QA2_RTOL_max_UVW:   tolerance_vector,
     GPS_QA2_RTOL_UVW_A:     tolerance_vector,
     GPS_QA2_RTOL_UVW_B:     tolerance_vector,
     GPS_QA2_RTOL_UVW_C:     tolerance_vector,
     GPS_QA2_VTOL_max_UVW:   tolerance_vector,
     GPS_QA2_VTOL_UVW_A:     tolerance_vector,
     GPS_QA2_VTOL_UVW_B:     tolerance_vector,
     GPS_QA2_VTOL_UVW_C:     tolerance_vector,
```

```
      GPS_QA3_RTOL:           nonzero_real,
      GPS_QA3_VTOL:           nonzero_real,
      M_WGS84_to_EF:          WGS84_to_EF_matrix,
      QA2_DELR_ratio_SF:      nonzero_real,
      QA2_DELV_ratio_SF:      nonzero_real,
      QA2_RTOL_SF:            nonzero_real,
      QA2_VTOL_SF:            nonzero_real,
      QA3_DELR_ratio_SF:      nonzero_real,
      QA3_DELV_ratio_SF:      nonzero_real,
      QA3_RTOL_SF:            nonzero_real,
      QA3_VTOL_SF:            nonzero_real,
      SF_pos:                 nonzero_real,
      SF_vel:                 nonzero_real,
      sig_diag_GPS_nom:       cov_diagonal_vector
      #]

rec_sp_K_loads: TYPE = [#
      acc_prop_min:           real,
      GPS_SW_cap:             num_GPS
      #]

rec_sp_constants: TYPE = [#
      ATFL_OV:                nat,
      deg_to_rad:             real,
      nautmi_per_ft:          real
      #]

rec_sp_outputs: TYPE = [#
      corr_coeff_GPS:         corr_coeff_vector,
      crew_des_rcvr_rcvd:     GPS_predicate,
      crew_QA_override_rcvd:  GPS_predicate,
      crew_QA_ovrd_status_ind: GPS_predicate,
      DELR_ratio_QA2_display: GPS_ratios,
      DELR_ratio_QA2_DL_2:    int,
      DELR_ratio_QA3_display: GPS_ratios,
      DELR_ratio_QA3_DL_2:    int,
      DELR_SV_sel:            real,
      delta_ratio_SV_sel:     real,
      delta_ratio_SV_sel_ind: display_indicator,
      delta_ratio_SV_sel_stat: display_status,
      DELV_ratio_QA2_display: GPS_ratios,
      DELV_ratio_QA2_DL_2:    int,
      DELV_ratio_QA3_display: GPS_ratios,
      DELV_ratio_QA3_DL_2:    int,
      DT_GPS_off_min:         delta_time,
      DT_QA2_DL:              int,
      FOM_nav:                GPS_FOM_vector,
      FOM_SV_sel:             GPS_figure_of_merit,
      G_ref_anncd_reset:      GPS_accelerations,
      GPS_anncd_reset:        GPS_predicate,
      GPS_anncd_reset_avail:  GPS_predicate,
      GPS_collection_word:    collection_word,
      GPS_collection_word_two: collection_word,
      GPS_DG_SF:              GPS_predicate,
      GPS_fail_FOM_disp:      GPS_predicate,
      GPS_fail_QA2_GAX:       GPS_predicate,
      GPS_fail_QA2_R:         GPS_predicate,
      GPS_fail_QA2_V:         GPS_predicate,
      GPS_fail_QA3_GAX:       GPS_predicate,
      GPS_fail_QA3_R:         GPS_predicate,
      GPS_fail_QA3_V:         GPS_predicate,
      GPS_off_alert:          bool,
      GPS_off_alert_GAX:      bool,
      GPS_QA2_RTOL_UVW_DL:    UVW_int_vector,
      GPS_QA2_VTOL_UVW_DL:    UVW_int_vector,
      GPS_sel_alt_disp:       distance,
      GPS_sel_alt_DL:         int,
      GPS_sel_DELR_UVW:       delta_position_vector_UVW,
      GPS_sel_DELR_UVW_DL:    UVW_int_vector,
      GPS_sel_DELV_UVW:       delta_velocity_vector_UVW,
      GPS_sel_DELV_UVW_DL:    UVW_int_vector,
      GPS_sel_lat_disp:       latitude,
      GPS_sel_lat_DL:         int,
      GPS_sel_lat_ind:        display_indicator,
      GPS_sel_lon_disp:       longitude,
      GPS_sel_lon_DL:         int,
      GPS_sel_lon_ind:        display_indicator,
      GPS_SV_sel_avail:       bool,
      num_GPS_sel:            nat,
      R_GPS_sel:              position_vector,
      R_GPS_sel_DL:           M50_int_vector,
      R_ref_anncd_reset:      GPS_positions,
      sel_cmd:                GPS_predicate,
      sel_rcvr:               GPS_predicate,
      sig_diag_GPS:           cov_diagonal_vector,
      T_GPS_sel:              mission_time,
      T_GPS_sel_DL:           real,
      T_ref_anncd_reset:      GPS_times,
      V_GPS_sel:              velocity_vector,
      V_GPS_sel_DL:           M50_int_vector,
      V_IMU_ref_anncd_reset:  GPS_velocities,
      V_ref_anncd_reset:      GPS_velocities
      #]

rec_sp_result: TYPE = [# output: rec_sp_outputs, state: rec_sp_state #]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal function: GPS Receiver State Processing
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GPS_receiver_state_processing(rec_sp_inputs:     rec_sp_inputs,
                              rec_sp_state:      rec_sp_state,
                              rec_sp_I_loads:    rec_sp_I_loads,
                              rec_sp_K_loads:    rec_sp_K_loads,
                              rec_sp_constants:  rec_sp_constants
                              ) : rec_sp_result =

  LET IMU_assign_out =
          IMU_assign(
              GPS_installed        (rec_sp_I_loads),
              GPS_SW_cap           (rec_sp_K_loads),
              nav_IMU_to_GPS       (rec_sp_inputs),
              V_current_filt       (rec_sp_inputs),
              V_last_GPS_two       (rec_sp_state) ),

      nav_state_prop_out =
          nav_state_propagation(
              G_two_prev           (rec_sp_state),
              GPS_DG               (rec_sp_inputs),
              GPS_DG_SF_prev       (rec_sp_state),
              GPS_IMU_vel_thresh   (rec_sp_I_loads),
              GPS_SV_sel_avail     (rec_sp_state),
              GPS_SW_cap           (rec_sp_K_loads),
              M_WGS84_to_EF        (rec_sp_I_loads),
              nav_MM_code          (rec_sp_inputs),
              R_GPS_ECEF           (rec_sp_inputs),
              R_GPS_prev           (rec_sp_state),
              R_GPS_sel            (rec_sp_state),
              T_current_filt       (rec_sp_inputs),
              T_GPS                (rec_sp_inputs),
              T_GPS_prev           (rec_sp_state),
              T_GPS_sel            (rec_sp_state),
              T_last_GPS           (rec_sp_inputs),
              V_current_filt       (rec_sp_inputs),
              V_current_GPS        (IMU_assign_out),
              V_GPS_ECEF           (rec_sp_inputs),
              V_GPS_prev           (rec_sp_state),
              V_GPS_sel            (rec_sp_state),
              V_last_GPS           (rec_sp_inputs),
              V_last_GPS_prev      (IMU_assign_out),
              V_last_GPS_sel       (rec_sp_state) ),

      SV_qual_assess_out =
          state_vector_quality_assessment(
              G_two                (nav_state_prop_out),
              GPS_DG_SF            (nav_state_prop_out),
              GPS_DG_SF_prev       (rec_sp_state),
              GPS_fail_FOM         (rec_sp_inputs),
              GPS_GAX_msg_cnt_min  (rec_sp_I_loads),
              GPS_QA2_RTOL_max_UVW (rec_sp_I_loads),
              GPS_QA2_RTOL_UVW_A   (rec_sp_I_loads),
              GPS_QA2_RTOL_UVW_B   (rec_sp_I_loads),
              GPS_QA2_RTOL_UVW_C   (rec_sp_I_loads),
              GPS_QA2_VTOL_max_UVW (rec_sp_I_loads),
              GPS_QA2_VTOL_UVW_A   (rec_sp_I_loads),
              GPS_QA2_VTOL_UVW_B   (rec_sp_I_loads),
              GPS_QA2_VTOL_UVW_C   (rec_sp_I_loads),
              GPS_QA3_RTOL         (rec_sp_I_loads),
              GPS_QA3_VTOL         (rec_sp_I_loads),
              GPS_SW_cap           (rec_sp_K_loads),
              max_values           (rec_sp_state),
              nav_reset            (rec_sp_inputs),
              noop_GPS_GAX_msgs    (rec_sp_inputs),
              QA2_fail_GAX_msg_cnt (rec_sp_state),
              QA3_fail_GAX_msg_cnt (rec_sp_state),
              R_filt               (rec_sp_inputs),
              R_GPS                (nav_state_prop_out),
              R_GPS_prev           (nav_state_prop_out),
              T_current_filt       (rec_sp_inputs),
              T_GPS                (nav_state_prop_out),
              T_GPS_last_update    (rec_sp_inputs),
              V_filt               (rec_sp_inputs),
              V_GPS                (nav_state_prop_out),
              V_GPS_prev           (nav_state_prop_out) ),

  state_vect_sel_out =
      state_vector_selection(
              corr_coeff_GPS_nom   (rec_sp_I_loads),
              crew_deselect_rcvr   (rec_sp_inputs),
              deg_to_rad           (rec_sp_constants),
              DELR_GPS_QA2_UVW     (SV_qual_assess_out),
              DELR_mag             (SV_qual_assess_out),
              delta_ratio_QA2_ind  (SV_qual_assess_out),
              delta_ratio_QA2_max  (SV_qual_assess_out),
              DELV_GPS_QA2_UVW     (SV_qual_assess_out),
              FOM                  (rec_sp_inputs),
              FOM_cov_SF_table     (rec_sp_I_loads),
              GPS_AIF_rcvd         (rec_sp_inputs),
              GPS_alert_DT         (rec_sp_I_loads),
              GPS_DG_SF            (nav_state_prop_out),
              GPS_fail_FOM         (rec_sp_inputs),
              GPS_fail_QA          (SV_qual_assess_out),
              GPS_GAX_msg_cnt_min  (rec_sp_I_loads),
              GPS_mode_rcvd        (rec_sp_inputs),
              GPS_QA2_RTOL_UVW     (SV_qual_assess_out),
              GPS_QA2_VTOL_UVW     (SV_qual_assess_out),
              GPS_SV_sel_avail     (rec_sp_state),
              GPS_SW_cap           (rec_sp_K_loads),
```

```
        M_M50_to_UVW          (SV_qual_assess_out),
        nautmi_per_ft         (rec_sp_constants),
        noop_GPS_GAX_msgs     (rec_sp_inputs),
        off_alert_GAX_msg_cnt (rec_sp_state),
        QA_override           (rec_sp_inputs),
        R_filt                (rec_sp_inputs),
        R_GPS                 (nav_state_prop_out),
        R_GPS_sel             (nav_state_prop_out),
        sig_diag_GPS_nom      (rec_sp_I_loads),
        T_current_filt        (rec_sp_inputs),
        T_GPS                 (nav_state_prop_out),
        T_GPS_off             (rec_sp_state),
        T_GPS_sel             (nav_state_prop_out),
        V_filt                (rec_sp_inputs),
        V_GPS                 (nav_state_prop_out),
        V_GPS_sel             (nav_state_prop_out) ),

ref_st_ann_reset_out =
    ref_state_announced_reset(
        DT_anncd_reset        (rec_sp_I_loads),
        G_two                 (nav_state_prop_out),
        GPS_DG_SF             (nav_state_prop_out),
        GPS_SW_cap            (rec_sp_K_loads),
        R_GPS                 (nav_state_prop_out),
        T_anncd_reset         (rec_sp_state),
        T_current_filt        (rec_sp_inputs),
        T_GPS                 (nav_state_prop_out),
        V_current_GPS         (IMU_assign_out),
        V_GPS                 (nav_state_prop_out) ),

GPS_downlist_out =
    GPS_downlist_computation(
        alt_SF                (rec_sp_I_loads),
        ang_SF                (rec_sp_I_loads),
        crew_deselect_rcvr    (rec_sp_inputs),
        DELR_ratio_QA2_max    (SV_qual_assess_out),
        DELR_ratio_QA3_display  (SV_qual_assess_out),
        DELR_UVW_pos_SF       (rec_sp_I_loads),
        DELV_ratio_QA2_max    (SV_qual_assess_out),
        DELV_ratio_QA3_display  (SV_qual_assess_out),
        DELV_UVW_vel_SF       (rec_sp_I_loads),
        DT_QA2                (SV_qual_assess_out),
        DT_QA2_SF             (rec_sp_I_loads),
        FOM_SV_sel            (state_vect_sel_out),
        GPS_AIF_rcvd          (rec_sp_inputs),
        GPS_DG_SF             (nav_state_prop_out),
        GPS_fail_FOM          (rec_sp_inputs),
        GPS_fail_QA           (SV_qual_assess_out),
        GPS_fail_QA2          (SV_qual_assess_out),
        GPS_fail_QA2_R        (SV_qual_assess_out),
        GPS_fail_QA2_V        (SV_qual_assess_out),
        GPS_fail_QA3          (SV_qual_assess_out),
        GPS_fail_QA3_R        (SV_qual_assess_out),
        GPS_fail_QA3_V        (SV_qual_assess_out),
        GPS_off_alert         (state_vect_sel_out),
        GPS_QA2_RTOL_UVW      (SV_qual_assess_out),
        GPS_QA2_VTOL_UVW      (SV_qual_assess_out),
        GPS_sel_alt           (state_vect_sel_out),
        GPS_sel_DELR_UVW      (state_vect_sel_out),
        GPS_sel_DELV_UVW      (state_vect_sel_out),
        GPS_sel_lat           (state_vect_sel_out),
        GPS_sel_lon           (state_vect_sel_out),
        GPS_to_nav_rcvd       (rec_sp_inputs),
        GPS_to_nav_force_rcvd (rec_sp_inputs),
        MVS_rcvr_sel          (state_vect_sel_out),
        nautmi_per_ft         (rec_sp_constants),
        QA2_DELR_ratio_SF     (rec_sp_I_loads),
        QA2_DELV_ratio_SF     (rec_sp_I_loads),
        QA2_RTOL_SF           (rec_sp_I_loads),
        QA2_VTOL_SF           (rec_sp_I_loads),
        QA3_DELR_ratio_SF     (rec_sp_I_loads),
        QA3_DELV_ratio_SF     (rec_sp_I_loads),
        R_GPS_sel             (state_vect_sel_out),
        SF_pos                (rec_sp_I_loads),
        SF_vel                (rec_sp_I_loads),
        T_GPS_sel             (state_vect_sel_out),
        V_GPS_sel             (state_vect_sel_out) )

IN (* output := (*
        corr_coeff_GPS     := corr_coeff_GPS     (state_vect_sel_out),
        crew_des_rcvr_rcvd := crew_des_rcvr_rcvd (state_vect_sel_out),
        crew_QA_override_rcvd :=
                crew_QA_override_rcvd         (state_vect_sel_out),
        crew_QA_ovrd_status_ind :=
                crew_QA_ovrd_status_ind       (state_vect_sel_out),
        DELR_ratio_QA2_display :=
                DELR_ratio_QA2_display        (SV_qual_assess_out),
        DELR_ratio_QA2_DL_2 := DELR_ratio_QA2_DL_2 (GPS_downlist_out),
        DELR_ratio_QA3_display :=
                DELR_ratio_QA3_display        (SV_qual_assess_out),
        DELR_ratio_QA3_DL_2 := DELR_ratio_QA3_DL_2 (GPS_downlist_out),
        DELR_SV_sel        := DELR_SV_sel        (state_vect_sel_out),
        delta_ratio_SV_sel := delta_ratio_SV_sel (state_vect_sel_out),
        delta_ratio_SV_sel_ind :=
                delta_ratio_SV_sel_ind        (state_vect_sel_out),
        delta_ratio_SV_sel_stat :=
                delta_ratio_SV_sel_stat       (state_vect_sel_out),
        DELV_ratio_QA2_display :=
                DELV_ratio_QA2_display        (SV_qual_assess_out),
        DELV_ratio_QA2_DL_2 := DELV_ratio_QA2_DL_2 (GPS_downlist_out),
```

```
DELV_ratio_QA3_display :=
          DELV_ratio_QA3_display        (SV_qual_assess_out),
DELV_ratio_QA3_DL_2 := DELV_ratio_QA3_DL_2 (GPS_downlist_out),
DT_GPS_off_min      := DT_GPS_off_min     (state_vect_sel_out),
DT_QA2_DL           := DT_QA2_DL          (GPS_downlist_out),
FOM_nav             := FOM_nav            (state_vect_sel_out),
FOM_SV_sel          := FOM_SV_sel         (state_vect_sel_out),
G_ref_anncd_reset   := G_ref_anncd_reset  (ref_st_ann_reset_out),
GPS_anncd_reset     := GPS_anncd_reset    (ref_st_ann_reset_out),
GPS_anncd_reset_avail :=
          GPS_anncd_reset_avail         (ref_st_ann_reset_out),
GPS_collection_word :=
          GPS_collection_word           (GPS_downlist_out),
GPS_collection_word_two :=
          GPS_collection_word_two       (GPS_downlist_out),
GPS_DG_SF           := GPS_DG_SF          (nav_state_prop_out),
GPS_fail_FOM_disp   := GPS_fail_FOM_disp  (state_vect_sel_out),
GPS_fail_QA2_GAX    := GPS_fail_QA2_GAX   (SV_qual_assess_out),
GPS_fail_QA2_R      := GPS_fail_QA2_R     (SV_qual_assess_out),
GPS_fail_QA2_V      := GPS_fail_QA2_V     (SV_qual_assess_out),
GPS_fail_QA3_GAX    := GPS_fail_QA3_GAX   (SV_qual_assess_out),
GPS_fail_QA3_R      := GPS_fail_QA3_R     (SV_qual_assess_out),
GPS_fail_QA3_V      := GPS_fail_QA3_V     (SV_qual_assess_out),
GPS_off_alert       := GPS_off_alert      (state_vect_sel_out),
GPS_off_alert_GAX   := GPS_off_alert_GAX  (state_vect_sel_out),
GPS_QA2_RTOL_UVW_DL :=
          GPS_QA2_RTOL_UVW_DL           (GPS_downlist_out),
GPS_QA2_VTOL_UVW_DL :=
          GPS_QA2_VTOL_UVW_DL           (GPS_downlist_out),
GPS_sel_alt_disp    := GPS_sel_alt_disp   (state_vect_sel_out),
GPS_sel_alt_DL      := GPS_sel_alt_DL     (GPS_downlist_out),
GPS_sel_DELR_UVW    := GPS_sel_DELR_UVW   (state_vect_sel_out),
GPS_sel_DELR_UVW_DL := GPS_sel_DELR_UVW_DL (GPS_downlist_out),
GPS_sel_DELV_UVW    := GPS_sel_DELV_UVW   (state_vect_sel_out),
GPS_sel_DELV_UVW_DL := GPS_sel_DELV_UVW_DL (GPS_downlist_out),
GPS_sel_lat_disp    := GPS_sel_lat_disp   (state_vect_sel_out),
GPS_sel_lat_DL      := GPS_sel_lat_DL     (GPS_downlist_out),
GPS_sel_lat_ind     := GPS_sel_lat_ind    (state_vect_sel_out),
GPS_sel_lon_disp    := GPS_sel_lon_disp   (state_vect_sel_out),
GPS_sel_lon_DL      := GPS_sel_lon_DL     (GPS_downlist_out),
GPS_sel_lon_ind     := GPS_sel_lon_ind    (state_vect_sel_out),
GPS_SV_sel_avail    := GPS_SV_sel_avail   (state_vect_sel_out),
num_GPS_sel         := num_GPS_sel        (state_vect_sel_out),
R_GPS_sel           := R_GPS_sel          (state_vect_sel_out),
R_GPS_sel_DL        := R_GPS_sel_DL       (GPS_downlist_out),
R_ref_anncd_reset   := R_ref_anncd_reset  (ref_st_ann_reset_out),
sel_cmd             := sel_cmd            (state_vect_sel_out),
sel_rcvr            := sel_rcvr           (state_vect_sel_out),
sig_diag_GPS        := sig_diag_GPS       (state_vect_sel_out),
T_GPS_sel           := T_GPS_sel          (state_vect_sel_out),
T_GPS_sel_DL        := T_GPS_sel_DL       (GPS_downlist_out),
T_ref_anncd_reset   := T_ref_anncd_reset  (ref_st_ann_reset_out),
V_GPS_sel           := V_GPS_sel          (state_vect_sel_out),
V_GPS_sel_DL        := V_GPS_sel_DL       (GPS_downlist_out),
V_IMU_ref_anncd_reset :=
          V_IMU_ref_anncd_reset         (ref_st_ann_reset_out),
V_ref_anncd_reset   := V_ref_anncd_reset  (ref_st_ann_reset_out)
#),

state := (#
  G_two_prev          := G_two_prev         (SV_qual_assess_out),
  GPS_DG_SF_prev      := GPS_DG_SF_prev     (SV_qual_assess_out),
  GPS_SV_sel_avail    := GPS_SV_sel_avail   (state_vect_sel_out),
  max_values          := max_values         (SV_qual_assess_out),
  off_alert_GAX_msg_cnt :=
          off_alert_GAX_msg_cnt         (state_vect_sel_out),
  QA2_fail_GAX_msg_cnt := QA2_fail_GAX_msg_cnt (SV_qual_assess_out),
  QA3_fail_GAX_msg_cnt := QA3_fail_GAX_msg_cnt (SV_qual_assess_out),
  R_GPS_prev          := R_GPS_prev         (SV_qual_assess_out),
  R_GPS_sel           := R_GPS_sel          (state_vect_sel_out),
  T_anncd_reset       := T_anncd_reset      (ref_st_ann_reset_out),
  T_GPS_off           := T_GPS_off          (state_vect_sel_out),
  T_GPS_prev          := T_GPS_prev         (SV_qual_assess_out),
  T_GPS_sel           := T_GPS_sel          (state_vect_sel_out),
  V_GPS_prev          := V_GPS_prev         (SV_qual_assess_out),
  V_GPS_sel           := V_GPS_sel          (state_vect_sel_out),
  V_last_GPS_sel      := V_last_GPS_sel     (nav_state_prop_out),
  V_last_GPS_two      := V_last_GPS_two     (nav_state_prop_out)
  #)
#)


END rec_sp_pf
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% -*- Mode: Pvs -*- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ref_state_proc.pvs --
%% Author          : Ben L. Di Vito
%% Created On      : Fri Apr 28 10:33:17 1995
%% Last Modified By: Ben L. Di Vito
%% Last Modified On: Wed May 22 09:46:34 1996
%% Update Count    : 72
%% Status          : Baselined
%%
%% HISTORY
%%   Originally Created On      : Fri Jun  3 11:05:42 1994
%%       for GPS CR 90810C
%%   Second version adapted for GPS CR 91051B, issued  3/20/95
%%   Updated to comply with     GPS CR 91051D, issued  6/09/95
%%   Updated to comply with     GPS CR 91051E, issued  6/23/95
%%   Updated to comply with     GPS CR 91051K, issued 10/30/95,
%%      along with misc. clean-up items
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal Function:  Reference State Processing (4.18) (new)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ref_sp_types: THEORY
BEGIN

%%%%%%%%
%% NOTE: PVS Boolean types are used to model Shuttle discrete types, whose
%%       constants are ON and OFF, as well as integer types restricted to the
%%       values 0 and 1.
%%%%%%%%


%%%%%%%%%%% Time types %%%%%%%%%%

mission_time:          TYPE = real

delta_time:            TYPE = real


%%%%%%%%%%% Index types %%%%%%%%%%

num_GPS:               TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

IMU_id:                TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

GPS_id:                TYPE = {n: nat | 1 <= n & n <= 3} CONTAINING 1

IMU_to_GPS_map:        TYPE = [GPS_id -> IMU_id]


%%%%%%%%%% Coordinate systems and position/velocity vectors %%%%%%%%%%

M50_axis:              TYPE = {Xm, Ym, Zm}

IMPORTING              vectors[M50_axis]

M50_vector:            TYPE = vector[M50_axis]

position_vector:       TYPE = M50_vector
velocity_vector:       TYPE = M50_vector
acceleration_vector:   TYPE = M50_vector

delta_position_vector: TYPE = M50_vector
delta_velocity_vector: TYPE = M50_vector

GPS_positions:         TYPE = [GPS_id -> position_vector]
GPS_velocities:        TYPE = [GPS_id -> velocity_vector]
GPS_accelerations:     TYPE = [GPS_id -> acceleration_vector]

GPS_delta_positions:   TYPE = [GPS_id -> delta_position_vector]
GPS_delta_velocities:  TYPE = [GPS_id -> delta_velocity_vector]

IMU_velocities:        TYPE = [IMU_id -> velocity_vector]


%%%%%%%%%% Modes and flags %%%%%%%%%%

AIF_flag:        TYPE = {auto, inhibit, force}

receiver_mode:   TYPE = {init, test, nav, blank}

nav_submode:     TYPE = {ins}   %% Other modes

major_mode_code: TYPE = nat     %% Fixed set of codes (not enumerated here)


%%%%%%%%%% Other GPS vectors %%%%%%%%%%

GPS_predicate:         TYPE = [GPS_id -> bool]

GPS_times:             TYPE = [GPS_id -> mission_time]

GPS_ratios:            TYPE = [GPS_id -> real]


GPS_rcvr_mode_vector: TYPE = [GPS_id -> receiver_mode]
```

```
GPS_nav_submodes:     TYPE = [GPS_id -> nav_submode]


%%%%%%%%%%% Default/null constants ("don't care" values) %%%%%%%%%%

null_mission_time:    mission_time
null_position:        position_vector
null_velocity:        velocity_vector
null_acceleration:    acceleration_vector

null_GPS_positions:   GPS_positions
null_GPS_velocities:  GPS_velocities


%%%%%%%%%%% Misc. constants, types, and operations %%%%%%%%%%


END ref_sp_types


%%%%%%%%%%%%%%%%%%%%%%%%%%  Subfunctions  %%%%%%%%%%%%%%%%%%%%%%%%%%

ref_sp_subfuns: THEORY
BEGIN

IMPORTING ref_sp_types


%%%%%%%%%%% Common variables %%%%%%%%%%

I:      VAR GPS_id
J:      VAR IMU_id


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Supporting functions (from common NAV subfunctions)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% Compute acceleration based on OPS-specific algorithm %%%%%%%%%%

accel_GPS(deg:     nat,
          ord:     nat,
          I_drag:  nat,
          I_vent:  nat,
          R:       position_vector,
          V:       velocity_vector,
          T:       mission_time) : acceleration_vector


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



```
%% Subfunction 4.6.4.2: GPS External Data Snap
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


ext_data_snap_out: TYPE = [#
      filt_restart_rcvd:  GPS_predicate,
      GPS_init_request:   GPS_predicate,
      GPS_mode:           GPS_rcvr_mode_vector,
      IMU_1_fail_GPS:     bool,
      IMU_2_fail_GPS:     bool,
      IMU_3_fail_GPS:     bool,
      T_IMU:              mission_time,
      V_IMU_M50:          GPS_velocities
      #]

external_data_snap(filt_restart_request:  GPS_predicate,
                   GPS_init_request:      GPS_predicate,
                   GPS_mode:              GPS_rcvr_mode_vector,
                   GPS_SW_cap:            num_GPS,
                   IMU_1_fail_GPS:        bool,
                   IMU_2_fail_GPS:        bool,
                   IMU_3_fail_GPS:        bool,
                   T_IMU:                 mission_time,
                   V_IMU_M50:             GPS_velocities
                   ) : ext_data_snap_out =

   (# filt_restart_rcvd := filt_restart_request,
      GPS_init_request  := GPS_init_request,
      GPS_mode          := GPS_mode,
      IMU_1_fail_GPS    := IMU_1_fail_GPS,
      IMU_2_fail_GPS    := IMU_2_fail_GPS,
      IMU_3_fail_GPS    := IMU_3_fail_GPS,
      T_IMU             := T_IMU,
      V_IMU_M50         := V_IMU_M50
    #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.6.4.3: IMU GPS Selection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


IMU_GPS_sel_out: TYPE = [#
      GPS_IMU_reconfig:  GPS_predicate,
      IMU_to_GPS:        IMU_to_GPS_map
      #]

null_IMU_GPS_sel_out: IMU_GPS_sel_out
```

```
IMU_GPS_selection : IMU_GPS_sel_out =

   null_IMU_GPS_sel_out WITH [(GPS_IMU_reconfig)(2) := false,
                              (IMU_to_GPS)(2) := 2]


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.6.4.4: GPS Reference State Initialization and Reset
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ref_init_reset_step_out: TYPE = [#
      aiding_data_valid:      bool,
      filt_restart_request:   bool,
      G_GPS:                  acceleration_vector,
      GPS_anncd_reset:        bool,
      GPS_anncd_reset_avail:  bool,
      GPS_anncd_reset_set:    bool,
      GPS_filt_restart:       bool,
      GPS_IMU_reconfig:       bool,
      GPS_init_request:       bool,
      IMU_reconfig_set:       bool,
      init_state_set:         bool,
      OPS_trans_GPS:          bool,
      R_ref:                  position_vector,
      T_IMU_prev:             mission_time,
      T_ref:                  mission_time,
      V_IMU_GPS:              velocity_vector,
      V_ref:                  velocity_vector
      #]

null_ref_init_reset_step_out: ref_init_reset_step_out


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.6.4.4.3: IMU to Receiver Reconfiguration and Announced Resets
%%                 for the Ith Receiver (substeps 1.1-1.3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ref_anncd_reset_step(init_vars:               ref_init_reset_step_out,
                     G_ref_anncd_reset_I:     acceleration_vector,
                     GPS_anncd_reset_avail_I: bool,
                     GPS_IMU_reconfig_I:      bool,
                     R_ref_anncd_reset_I:     position_vector,
                     T_ref_anncd_reset_I:     mission_time,
                     V_IMU_ref_anncd_reset_J: velocity_vector,
                     V_ref_anncd_reset_I:     velocity_vector
                     ) : ref_init_reset_step_out =

  LET anncd_vars =
        init_vars WITH [(R_ref)       := R_ref_anncd_reset_I,
                        (V_ref)       := V_ref_anncd_reset_I,
                        (T_ref)       := T_ref_anncd_reset_I,
                        (T_IMU_prev)  := T_ref_anncd_reset_I,
                        (V_IMU_GPS)   := V_IMU_ref_anncd_reset_J,
                        (G_GPS)       := G_ref_anncd_reset_I ],
      anncd_vars_2 =
        IF GPS_anncd_reset_avail_I AND GPS_IMU_reconfig_I
           THEN anncd_vars WITH [(IMU_reconfig_set)    := true,
                                 (GPS_IMU_reconfig)    := false ]
           ELSE anncd_vars WITH [(GPS_anncd_reset_set) := true,
                                 (GPS_anncd_reset)     := false ]
        ENDIF

    IN  anncd_vars_2 WITH [(GPS_anncd_reset_avail) := false]


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Step 4.6.4.4.1: Reference State Initialization and Reset (2.4.1-2.4.3)
%% Step 4.6.4.4.2: OPS Transition Filter Restarts and Filter Restarts
%%                 During the OPS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ref_init_reset_step(I:                        GPS_id,
                    J:                        IMU_id,
                    aiding_data_valid:        GPS_predicate,
                    filt_restart_request_rcvd: GPS_predicate,
                    G_GPS:                    GPS_accelerations,
                    G_GPS_init:               acceleration_vector,
                    G_ref_anncd_reset:        GPS_accelerations,
                    GPS_anncd_reset:          GPS_predicate,
                    GPS_anncd_reset_avail:    GPS_predicate,
                    GPS_anncd_reset_set:      GPS_predicate,
                    GPS_filt_restart:         GPS_predicate,
                    GPS_IMU_reconfig:         GPS_predicate,
                    GPS_init_request:         GPS_predicate,
                    GPS_nav_submode_rcvd:     GPS_nav_submodes,
                    IMU_reconfig_set:         GPS_predicate,
                    init_state_set:           GPS_predicate,
                    OPS_trans_GPS:            GPS_predicate,
                    R_ref_anncd_reset:        GPS_positions,
                    R_ref_init:               position_vector,
                    T_ref_anncd_reset:        GPS_times,
                    T_ref_init:               mission_time,
                    V_IMU_GPS_in:             IMU_velocities,
                    V_IMU_ref_anncd_reset:    IMU_velocities,
                    V_ref_anncd_reset:        GPS_velocities,
                    V_ref_init:               velocity_vector
                    ) : ref_init_reset_step_out =

  LET flags = null_ref_init_reset_step_out WITH [
```

```
            (aiding_data_valid)      := aiding_data_valid(I),
            (filt_restart_request)   := filt_restart_request_rcvd(I),
            (G_GPS)                  := G_GPS(I),
            (GPS_anncd_reset)        := GPS_anncd_reset(I),
            (GPS_anncd_reset_avail)  := GPS_anncd_reset_avail(I),
            (GPS_anncd_reset_set)    := GPS_anncd_reset_set(I),
            (GPS_filt_restart)       := GPS_filt_restart(I),
            (GPS_IMU_reconfig)       := GPS_IMU_reconfig(I),
            (GPS_init_request)       := GPS_init_request(I),
            (IMU_reconfig_set)       := IMU_reconfig_set(I),
            (init_state_set)         := init_state_set(I),
            (OPS_trans_GPS)          := OPS_trans_GPS(I) ],

   init_vars = IF GPS_init_request(I) OR
                  (OPS_trans_GPS(I) AND NOT aiding_data_valid(I)) OR
                  (filt_restart_request_rcvd(I) AND
                   GPS_nav_submode_rcvd(I) = ins)
                  THEN flags WITH [
                        (R_ref)             := R_ref_init,
                        (V_ref)             := V_ref_init,
                        (T_ref)             := T_ref_init,
                        (T_IMU_prev)        := T_ref_init,
                        (V_IMU_GPS)         := V_IMU_GPS_in(J),
                        (G_GPS)             := G_GPS_init,
                        (aiding_data_valid) := true ]
                  ELSE flags
               ENDIF
IN
   IF GPS_init_request(I)
      THEN init_vars WITH [(init_state_set)       := true,
                           (GPS_init_request)     := false,
                           (GPS_anncd_reset)      := false,
                           (GPS_IMU_reconfig)     := false ]
   ELSIF GPS_nav_submode_rcvd(I) /= ins
      THEN init_vars WITH [(GPS_anncd_reset)      := false,
                           (GPS_IMU_reconfig)     := false ]
   ELSIF OPS_trans_GPS(I)
      THEN init_vars WITH [(filt_restart_request) := true,
                           (OPS_trans_GPS)        := false ]
   ELSIF filt_restart_request_rcvd(I)
      THEN init_vars WITH [(GPS_filt_restart)     := true,
                           (filt_restart_request) := false,
                           (GPS_anncd_reset)      := false ]
   ELSIF (GPS_anncd_reset_avail(I) AND GPS_IMU_reconfig(I)) OR
         GPS_anncd_reset(I)
      THEN ref_anncd_reset_step(init_vars,
                                G_ref_anncd_reset(I),
                                GPS_anncd_reset_avail(I),
                                GPS_IMU_reconfig(I),
                                R_ref_anncd_reset(I),
                                T_ref_anncd_reset(I),
                                V_IMU_ref_anncd_reset(J),
                                V_ref_anncd_reset(I))
      ELSE init_vars
   ENDIF


%%%%%%%%%% Subfunction top-level specification %%%%%%%%%%


ref_state_init_reset_out: TYPE = [#
      aiding_data_valid:      GPS_predicate,
      filt_restart_request:   GPS_predicate,
      G_GPS:                  GPS_accelerations,          %%% state var
      GPS_anncd_reset:        GPS_predicate,
      GPS_anncd_reset_avail:  GPS_predicate,
      GPS_anncd_reset_set:    GPS_predicate,
      GPS_filt_restart:       GPS_predicate,
      GPS_IMU_reconfig:       GPS_predicate,              %%% Note 1
      GPS_init_request:       GPS_predicate,
      GPS_ref_init_avail:     bool,
      IMU_reconfig_set:       GPS_predicate,
      init_state_set:         GPS_predicate,
      OPS_trans_GPS:          GPS_predicate,
      R_ref:                  GPS_positions,
      T_IMU_prev:             GPS_times,                  %%% state var
      T_ref:                  GPS_times,
      V_IMU_GPS:              GPS_velocities,             %%% state var
      V_ref:                  GPS_velocities
      #]

%% Note 1: This variable is set by the IMU GPS Selection subfunction,
%% and is updated by the Reference State Initialization and Reset
%% subfunction.  It is not listed in the principal function output
%% table nor used by any other subfunction.

ref_state_init_reset(
        aiding_data_valid:          GPS_predicate,
        filt_restart_request_rcvd:  GPS_predicate,
        G_GPS:                      GPS_accelerations,  %%% state var
        G_GPS_init:                 acceleration_vector,
        G_ref_anncd_reset:          GPS_accelerations,
        GPS_anncd_reset:            GPS_predicate,
        GPS_anncd_reset_avail:      GPS_predicate,
        GPS_anncd_reset_set:        GPS_predicate,
        GPS_filt_restart:           GPS_predicate,
        GPS_IMU_reconfig:           GPS_predicate,
        GPS_init_request:           GPS_predicate,
        GPS_installed:              GPS_predicate,
        GPS_mode:                   GPS_rcvr_mode_vector,
```

```
    GPS_nav_submode_rcvd:       GPS_nav_submodes,
    GPS_ref_init_avail:         bool,
    GPS_SW_cap:                 num_GPS,
    IMU_reconfig_set:           GPS_predicate,
    IMU_to_GPS:                 IMU_to_GPS_map,
    init_state_set:             GPS_predicate,
    OPS_trans_GPS:              GPS_predicate,
    R_GPS_init:                 position_vector,
    R_ref_anncd_reset:          GPS_positions,
    T_GPS_init:                 mission_time,
    T_IMU_prev:                 GPS_times,          %%% state var
    T_ref_anncd_reset:          GPS_times,
    V_GPS_init:                 velocity_vector,
    V_IMU_GPS:                  GPS_velocities,     %%% state var
    V_IMU_GPS_init:             IMU_velocities,
    V_IMU_ref_anncd_reset:      IMU_velocities,
    V_ref_anncd_reset:          GPS_velocities
    ) : ref_state_init_reset_out =

LET R_ref_init =
      IF GPS_ref_init_avail THEN R_GPS_init ELSE null_position ENDIF,
    V_ref_init =
      IF GPS_ref_init_avail THEN V_GPS_init ELSE null_velocity ENDIF,
    T_ref_init =
      IF GPS_ref_init_avail THEN T_GPS_init ELSE null_mission_time ENDIF,
    G_ref_init =
      IF GPS_ref_init_avail THEN G_GPS_init ELSE null_acceleration ENDIF,
    V_IMU_GPS_in = IF GPS_ref_init_avail
                     THEN V_IMU_GPS_init
                     ELSE null_GPS_velocities
                   ENDIF,

    init_reset =
      (LAMBDA I:
        IF NOT GPS_installed(I)
           THEN null_ref_init_reset_step_out
        ELSIF GPS_mode(I) = blank
           THEN null_ref_init_reset_step_out
                  WITH [(aiding_data_valid) := false]
        ELSIF GPS_filt_restart(I) AND init_state_set(I) AND
              GPS_anncd_reset_set(I) AND IMU_reconfig_set(I)
           THEN ref_init_reset_step(I,
                                    IMU_to_GPS(I),
                                    aiding_data_valid,
                                    filt_restart_request_rcvd,
                                    G_GPS,
                                    G_GPS_init,
                                    G_ref_anncd_reset,
                                    GPS_anncd_reset,
                                    GPS_anncd_reset_avail,
                                    GPS_anncd_reset_set,
                                    GPS_filt_restart,
                                    GPS_IMU_reconfig,
                                    GPS_init_request,
                                    GPS_nav_submode_rcvd,
                                    IMU_reconfig_set,
                                    init_state_set,
                                    OPS_trans_GPS,
                                    R_ref_anncd_reset,
                                    R_ref_init,
                                    T_ref_anncd_reset,
                                    T_ref_init,
                                    V_IMU_GPS_in,
                                    V_IMU_ref_anncd_reset,
                                    V_ref_anncd_reset,
                                    V_ref_init)
           ELSE null_ref_init_reset_step_out
        ENDIF)

  IN (# aiding_data_valid     := (LAMBDA I: aiding_data_valid(init_reset(I))),
       filt_restart_request   :=
         (LAMBDA I: filt_restart_request(init_reset(I))),
       G_GPS                  := (LAMBDA I: G_GPS(init_reset(I))),
       GPS_anncd_reset        := (LAMBDA I: GPS_anncd_reset(init_reset(I))),
       GPS_anncd_reset_avail :=
         (LAMBDA I: GPS_anncd_reset_avail(init_reset(I))),
       GPS_anncd_reset_set    :=
         (LAMBDA I: GPS_anncd_reset_set(init_reset(I))),
       GPS_filt_restart       := (LAMBDA I: GPS_filt_restart(init_reset(I))),
       GPS_IMU_reconfig       := (LAMBDA I: GPS_IMU_reconfig(init_reset(I))),
       GPS_init_request       := (LAMBDA I: GPS_init_request(init_reset(I))),
       GPS_ref_init_avail     := false,
       IMU_reconfig_set       := (LAMBDA I: IMU_reconfig_set(init_reset(I))),
       init_state_set         := (LAMBDA I: init_state_set(init_reset(I))),
       OPS_trans_GPS          := (LAMBDA I: OPS_trans_GPS(init_reset(I))),
       R_ref                  := (LAMBDA I: R_ref(init_reset(I))),
       T_IMU_prev             := (LAMBDA I: T_IMU_prev(init_reset(I))),
       T_ref                  := (LAMBDA I: T_ref(init_reset(I))),
       V_IMU_GPS              := (LAMBDA I: V_IMU_GPS(init_reset(I))),
       V_ref                  := (LAMBDA I: V_ref(init_reset(I)))
     #)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subfunction 4.6.4.5: GPS Reference State_Propagation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prop_step_out: TYPE = [#
      G_GPS:      acceleration_vector,
```

```
      R_prop:     position_vector,
      T_IMU_prev: mission_time,
      T_prop:     mission_time,
      V_IMU_GPS:  velocity_vector,
      V_prop:     velocity_vector
      #]


ref_state_prop_step(I:                  GPS_id,
                    T_current_GPS_ref:  mission_time,
                    acc_thresh_GPS_ref: real,
                    G_GPS:              GPS_accelerations,
                    GM_deg_GPS_aid:     nat,
                    GM_ord_GPS_aid:     nat,
                    I_drag_GPS_ref:     nat,
                    I_vent_GPS_ref:     nat,
                    IMU_to_GPS:         IMU_to_GPS_map,
                    R_ref:              GPS_positions,
                    T_current:          mission_time,
                    T_IMU:              mission_time,
                    T_IMU_prev:         GPS_times,
                    T_ref:              GPS_times,
                    V_IMU_GPS:          GPS_velocities,
                    V_IMU_M50:          GPS_velocities,
                    V_ref:              GPS_velocities
                    ) : prop_step_out =

  LET J           = IMU_to_GPS(I),
      acc_IMU_raw = IF T_IMU = T_IMU_prev(I)
                      THEN zero_vector
                      ELSE scalar_mult(1 / (T_IMU - T_IMU_prev(I)),
                                       vector_diff(V_IMU_M50(J),
                                                   V_IMU_GPS(I)))
                    ENDIF,
      acc_IMU     = IF vector_mag(acc_IMU_raw) < acc_thresh_GPS_ref
                      THEN zero_vector
                      ELSE acc_IMU_raw
                    ENDIF,
      I_drag      = IF vector_mag(acc_IMU_raw) < acc_thresh_GPS_ref
                      THEN I_drag_GPS_ref
                      ELSE 0
                    ENDIF,
      I_vent      = IF vector_mag(acc_IMU_raw) < acc_thresh_GPS_ref
                      THEN I_vent_GPS_ref
                      ELSE 0
                    ENDIF,
      V_IMU_GPS_I = IF T_IMU = T_IMU_prev(I)
                      THEN V_IMU_GPS(I)
                      ELSE V_IMU_M50(J)
                    ENDIF,
      T_IMU_prev_I = T_IMU,

      DT_IMU  = T_current_GPS_ref - T_ref(I),
      T_ref_I = T_current_GPS_ref,

      DV_prop = scalar_mult(DT_IMU, acc_IMU),

      R_ref_I = vector_sum(
                  R_ref(I),
                  scalar_mult(
                    DT_IMU,
                    vector_sum(
                      V_ref(I),
                      scalar_mult(1/2,
                                  vector_sum(DV_prop,
                                             scalar_mult(DT_IMU,
                                                         G_GPS(I)))))),
      G_GPS_new = accel_GPS(GM_deg_GPS_aid, GM_ord_GPS_aid,
                            I_drag, I_vent,
                            R_ref_I, V_ref(I), T_ref_I),
      V_ref_I   = vector_sum(V_ref(I),
                             vector_sum(DV_prop,
                                        scalar_mult(DT_IMU / 2,
                                                    vector_sum(G_GPS(I),
                                                               G_GPS_new))))

    IN (# G_GPS      := G_GPS_new,
          R_prop     := R_ref_I,
          T_IMU_prev := T_IMU_prev_I,
          T_prop     := T_ref_I,
          V_IMU_GPS  := V_IMU_GPS_I,
          V_prop     := V_ref_I
        #)


%%%%%%%%%% Subfunction top-level specification %%%%%%%%%%


ref_state_prop_out: TYPE = [#
      G_GPS:      GPS_accelerations,      %%% state var
      R_ref:      GPS_positions,
      T_IMU_prev: GPS_times,              %%% state var
      T_ref:      GPS_times,
      V_IMU_GPS:  GPS_velocities,         %%% state var
      V_ref:      GPS_velocities
      #]


ref_state_propagation(acc_thresh_GPS_ref:  real,
                      aiding_data_valid:   GPS_predicate,
```

```
                G_GPS:              GPS_accelerations,  %%% state var
                GM_deg_GPS_aid:     nat,
                GM_ord_GPS_aid:     nat,
                GPS_installed:      GPS_predicate,
                GPS_SW_cap:         num_GPS,
                I_drag_GPS_ref:     nat,
                I_vent_GPS_ref:     nat,
                IMU_to_GPS:         IMU_to_GPS_map,
                R_ref:              GPS_positions,
                T_current:          mission_time,
                T_IMU:              mission_time,
                T_IMU_prev:         GPS_times,          %%% state var
                T_ref:              GPS_times,
                V_IMU_GPS:          GPS_velocities,     %%% state var
                V_IMU_M50:          GPS_velocities,
                V_ref:              GPS_velocities
                ) : ref_state_prop_out =

LET prop_state =
      (LAMBDA I:
         IF aiding_data_valid(I)
            THEN ref_state_prop_step(
                           I,
                           T_current,
                           acc_thresh_GPS_ref,
                           G_GPS,
                           GM_deg_GPS_aid,
                           GM_ord_GPS_aid,
                           I_drag_GPS_ref,
                           I_vent_GPS_ref,
                           IMU_to_GPS,
                           R_ref,
                           T_current,
                           T_IMU,
                           T_IMU_prev,
                           T_ref,
                           V_IMU_GPS,
                           V_IMU_M50,
                           V_ref)
            ELSE (# G_GPS      := G_GPS(I),
                    R_prop     := R_ref(I),
                    T_IMU_prev := T_IMU_prev(I),
                    T_prop     := T_ref(I),
                    V_IMU_GPS  := V_IMU_GPS(I),
                    V_prop     := V_ref(I)
                  #)
         ENDIF)

IN (# G_GPS      := (LAMBDA I: G_GPS(prop_state(I))),
      R_ref      := (LAMBDA I: R_prop(prop_state(I))),
      T_IMU_prev := (LAMBDA I: T_IMU_prev(prop_state(I))),
      T_ref      := (LAMBDA I: T_prop(prop_state(I))),
      V_IMU_GPS  := (LAMBDA I: V_IMU_GPS(prop_state(I))),
      V_ref      := (LAMBDA I: V_prop(prop_state(I)))
   #)



END ref_sp_subfuns


%%%%%%%%%%%%%%%%%%%%%%%%%  Principal Function  %%%%%%%%%%%%%%%%%%%%%%%%%

ref_sp_pf: THEORY
BEGIN

IMPORTING ref_sp_types, ref_sp_subfuns

ref_sp_inputs: TYPE = [#
      clock_clocktime:        mission_time,
      filt_restart_request:   GPS_predicate,
      G_GPS_init:             acceleration_vector,
      G_ref_anncd_reset:      GPS_accelerations,
      GPS_anncd_reset:        GPS_predicate,
      GPS_anncd_reset_avail:  GPS_predicate,
      GPS_anncd_reset_set:    GPS_predicate,
      GPS_filt_restart:       GPS_predicate,
      GPS_init_request:       GPS_predicate,
      GPS_mode_current:       GPS_rcvr_mode_vector,
      GPS_nav_submode_rcvd:   GPS_nav_submodes,
      GPS_ref_init_avail:     bool,
      IMU_1_fail:             bool,
      IMU_2_fail:             bool,
      IMU_3_fail:             bool,
      IMU_reconfig_set:       GPS_predicate,
      init_state_set:         GPS_predicate,
      OPS_trans_GPS:          GPS_predicate,
      R_avgg:                 position_vector,
      R_GPS_init:             position_vector,
      R_ref_anncd_reset:      GPS_positions,
      sel_vel_GPS_M50:        acceleration_vector,
      T_GPS_init:             mission_time,
      T_ref_anncd_reset:      GPS_times,
      T_state:                mission_time,
      time_V:                 mission_time,
      V_avgg:                 velocity_vector,
      V_GPS_init:             velocity_vector,
      V_IMU_GPS_init:         IMU_velocities,
      V_IMU_ref_anncd_reset:  IMU_velocities,
      V_ref_anncd_reset:      GPS_velocities
```

```
      #]

ref_sp_state: TYPE = [#
      aiding_data_valid:      GPS_predicate,              %%% Note 2
      G_GPS:                  GPS_accelerations,
      T_IMU_prev:             GPS_times,
      V_IMU_GPS:              GPS_velocities
      #]

%% Note 2: This variable is listed in the principal function output
%% table, but its previous value is also read by the principal function.
%% A state variable slot is therefore added to model this situation.

ref_sp_I_loads: TYPE = [#
      acc_thresh_GPS_ref:     real,
      GPS_installed:          GPS_predicate,   % Elements 1,3 are K-loads
      I_drag_GPS_ref:         nat,
      I_vent_GPS_ref:         nat
      #]

ref_sp_K_loads: TYPE = [#
      GM_deg_GPS_aid:         nat,
      GM_ord_GPS_aid:         nat,
      GPS_SW_cap:             num_GPS
      #]

ref_sp_constants: TYPE = [#
      ATFL_OV:                nat
      #]

ref_sp_outputs: TYPE = [#
      aiding_data_valid:      GPS_predicate,
      filt_restart_request:   GPS_predicate,
      GPS_anncd_reset:        GPS_predicate,      %%% Note 3
      GPS_anncd_reset_avail:  GPS_predicate,      %%% Note 3
      GPS_anncd_reset_set:    GPS_predicate,
      GPS_filt_restart:       GPS_predicate,
      GPS_init_request:       GPS_predicate,
      GPS_ref_init_avail:     bool,               %%% Note 3
      IMU_reconfig_set:       GPS_predicate,
      IMU_to_GPS:             IMU_to_GPS_map,
      init_state_set:         GPS_predicate,
      OPS_trans_GPS:          GPS_predicate,      %%% Note 3
      R_ref:                  GPS_positions,
      T_ref:                  GPS_times,
      V_ref:                  GPS_velocities
      #]

%% Note 3: These variables are not listed in the principal function
%% output table, but are added as pseudo-outputs to model the effect
%% of resetting these flags.

ref_sp_result: TYPE = [# output: ref_sp_outputs, state: ref_sp_state #]


%%%%%%%%%% Determine current OPS (replace with actual method) %%%%%%%%%%

current_OPS((ref_sp_inputs: ref_sp_inputs)): nat


%%%%%%%%%% Common variables %%%%%%%%%%

I:      VAR GPS_id


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  GPS Reference State Processing  (OPS 2, 3, 8)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GPS_reference_state_processing_1(ref_sp_inputs:    ref_sp_inputs,
                                 ref_sp_state:     ref_sp_state,
                                 ref_sp_I_loads:   ref_sp_I_loads,
                                 ref_sp_K_loads:   ref_sp_K_loads,
                                 ref_sp_constants: ref_sp_constants
                                 ) : ref_sp_result =

  LET ext_data_snap_out =
          external_data_snap(
              filt_restart_request  (ref_sp_inputs),
              GPS_init_request      (ref_sp_inputs),
              GPS_mode_current      (ref_sp_inputs),
              GPS_SW_cap            (ref_sp_K_loads),
              IMU_1_fail            (ref_sp_inputs),
              IMU_2_fail            (ref_sp_inputs),
              IMU_3_fail            (ref_sp_inputs),
              time_V                (ref_sp_inputs),
              (LAMBDA I: sel_vel_GPS_M50(ref_sp_inputs)) ),

      IMU_GPS_sel_out =  IMU_GPS_selection,

      ref_init_reset_out =
          ref_state_init_reset(
              aiding_data_valid     (ref_sp_state),
              filt_restart_rcvd     (ext_data_snap_out),
              G_GPS                 (ref_sp_state),
              G_GPS_init            (ref_sp_inputs),
              G_ref_anncd_reset     (ref_sp_inputs),
              GPS_anncd_reset       (ref_sp_inputs),
              GPS_anncd_reset_avail (ref_sp_inputs),
              GPS_anncd_reset_set   (ref_sp_inputs),
```

```
            GPS_filt_restart      (ref_sp_inputs),
            GPS_IMU_reconfig      (IMU_GPS_sel_out),
            GPS_init_request      (ext_data_snap_out),
            GPS_installed         (ref_sp_I_loads),
            GPS_mode              (ext_data_snap_out),
            GPS_nav_submode_rcvd  (ref_sp_inputs),
            GPS_ref_init_avail    (ref_sp_inputs),
            GPS_SW_cap            (ref_sp_K_loads),
            IMU_reconfig_set      (ref_sp_inputs),
            IMU_to_GPS            (IMU_GPS_sel_out),
            init_state_set        (ref_sp_inputs),
            OPS_trans_GPS         (ref_sp_inputs),
            R_GPS_init            (ref_sp_inputs),
            R_ref_anncd_reset     (ref_sp_inputs),
            T_GPS_init            (ref_sp_inputs),
            T_IMU_prev            (ref_sp_state),
            T_ref_anncd_reset     (ref_sp_inputs),
            V_GPS_init            (ref_sp_inputs),
            V_IMU_GPS             (ref_sp_state),
            V_IMU_GPS_init        (ref_sp_inputs),
            V_IMU_ref_anncd_reset (ref_sp_inputs),
            V_ref_anncd_reset     (ref_sp_inputs) ),

    ref_state_prop_out =
        ref_state_propagation(
            acc_thresh_GPS_ref    (ref_sp_I_loads),
            aiding_data_valid     (ref_init_reset_out),
            G_GPS                 (ref_init_reset_out),
            GM_deg_GPS_aid        (ref_sp_K_loads),
            GM_ord_GPS_aid        (ref_sp_K_loads),
            GPS_installed         (ref_sp_I_loads),
            GPS_SW_cap            (ref_sp_K_loads),
            I_drag_GPS_ref        (ref_sp_I_loads),
            I_vent_GPS_ref        (ref_sp_I_loads),
            IMU_to_GPS            (IMU_GPS_sel_out),
            R_ref                 (ref_init_reset_out),
            clock_clocktime       (ref_sp_inputs),
            T_IMU                 (ext_data_snap_out),
            T_IMU_prev            (ref_init_reset_out),
            T_ref                 (ref_init_reset_out),
            V_IMU_GPS             (ref_init_reset_out),
            V_IMU_M50             (ext_data_snap_out),
            V_ref                 (ref_init_reset_out) )
IN
   (# output := (#
        aiding_data_valid    := aiding_data_valid    (ref_init_reset_out),
        filt_restart_request := filt_restart_request (ref_init_reset_out),
        GPS_anncd_reset      := GPS_anncd_reset      (ref_init_reset_out),
        GPS_anncd_reset_avail := GPS_anncd_reset_avail (ref_init_reset_out),
        GPS_anncd_reset_set  := GPS_anncd_reset_set  (ref_init_reset_out),
        GPS_filt_restart     := GPS_filt_restart     (ref_init_reset_out),
        GPS_init_request     := GPS_init_request     (ref_init_reset_out),
        GPS_ref_init_avail   := GPS_ref_init_avail   (ref_init_reset_out),
        IMU_reconfig_set     := IMU_reconfig_set     (ref_init_reset_out),
        IMU_to_GPS           := IMU_to_GPS           (IMU_GPS_sel_out),
        init_state_set       := init_state_set       (ref_init_reset_out),
        OPS_trans_GPS        := OPS_trans_GPS        (ref_init_reset_out),
        R_ref                := R_ref                (ref_state_prop_out),
        T_ref                := T_ref                (ref_state_prop_out),
        V_ref                := V_ref                (ref_state_prop_out)
        #),

      state := (#
        aiding_data_valid    := aiding_data_valid    (ref_init_reset_out),
        G_GPS                := G_GPS                (ref_state_prop_out),
        T_IMU_prev           := T_IMU_prev           (ref_state_prop_out),
        V_IMU_GPS            := V_IMU_GPS            (ref_state_prop_out)
        #)
    #)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  GPS Reference State Processing (OPS 1, 6)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GPS_reference_state_processing_2(ref_sp_inputs:    ref_sp_inputs,
                                 ref_sp_state:     ref_sp_state,
                                 ref_sp_I_loads:   ref_sp_I_loads,
                                 ref_sp_K_loads:   ref_sp_K_loads,
                                 ref_sp_constants: ref_sp_constants
                                 ) : ref_sp_result =

  LET GPS_init_request = GPS_init_request (ref_sp_inputs),   %% SNAP

      R_ref = (LAMBDA I:
                 IF I = 2 THEN R_avgg(ref_sp_inputs) ELSE null_position ENDIF),
      V_ref = (LAMBDA I:
                 IF I = 2 THEN V_avgg(ref_sp_inputs) ELSE null_velocity ENDIF),
      T_ref = (LAMBDA I:
                 IF I=2 THEN T_state(ref_sp_inputs) ELSE null_mission_time ENDIF)

  IN
     (# output := (#
         aiding_data_valid    := (LAMBDA I: I = 2),
         filt_restart_request := filt_restart_request (ref_sp_inputs),
         GPS_anncd_reset      := GPS_anncd_reset      (ref_sp_inputs),
         GPS_anncd_reset_avail := GPS_anncd_reset_avail (ref_sp_inputs),
         GPS_anncd_reset_set  := GPS_anncd_reset_set  (ref_sp_inputs),
         GPS_filt_restart     := (LAMBDA I: I = 2 AND
                                        OPS_trans_GPS(ref_sp_inputs)(2)),
```

```
        GPS_init_request     := (LAMBDA I: false),
        GPS_ref_init_avail   := GPS_ref_init_avail   (ref_sp_inputs),
        IMU_reconfig_set     := IMU_reconfig_set     (ref_sp_inputs),
        IMU_to_GPS           := (LAMBDA I: 2),       %% not explicitly set
        init_state_set       := (LAMBDA I: I = 2 AND GPS_init_request(2)),
        OPS_trans_GPS        := (LAMBDA I: false),
        R_ref                := R_ref,
        T_ref                := T_ref,
        V_ref                := V_ref
        #),

      state := ref_sp_state WITH [
        aiding_data_valid    := (LAMBDA I: I = 2) ]
    #)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Principal function: GPS Reference State Processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GPS_reference_state_processing(ref_sp_inputs:    ref_sp_inputs,
                               ref_sp_state:     ref_sp_state,
                               ref_sp_I_loads:   ref_sp_I_loads,
                               ref_sp_K_loads:   ref_sp_K_loads,
                               ref_sp_constants: ref_sp_constants
                               ) : ref_sp_result =

  LET OPS = current_OPS(ref_sp_inputs)

  IN  IF OPS = 2 OR OPS = 8 OR OPS = 3
        THEN GPS_reference_state_processing_1(ref_sp_inputs,
                                              ref_sp_state,
                                              ref_sp_I_loads,
                                              ref_sp_K_loads,
                                              ref_sp_constants)
        ELSE GPS_reference_state_processing_2(ref_sp_inputs,
                                              ref_sp_state,
                                              ref_sp_I_loads,
                                              ref_sp_K_loads,
                                              ref_sp_constants)
      ENDIF


END ref_sp_pf
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1996	Contractor Report

4. TITLE AND SUBTITLE

Using Formal Methods to Assist in the Requirements Analysis of the Space Shuttle GPS Change **Request**

5. FUNDING NUMBERS

C NAS1-19341
C NAS9-18817
WU 323-08-01-02

6. AUTHOR(S)

Ben L. Divito
Larry W. Roberts

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

ViGYAN, Inc.
Hampton, VA **23666**

Lockheed Martin Space Information Systems
Houston, TX

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Adminsitration
Langley Research Center
Hampton, VA 23681-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

NASA CR-4752

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Ricky W. Butler
Final Report

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category 62

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

We describe a recent NASA-sponsored pilot project intended to gauge the effectiveness of using formal methods in Space Shuttle software requirements analysis. Several Change Requests (CRs) were selected as promising targets to demonstrate the utility of formal methods in this application domain. A CR to add new navigation capabilities to the Shuttle, based on Global Positioning System (GPS) technology, is the focus of this report.

Carried out in parallel with the Shuttle program's conventional requirements analysis process was a limited form of analysis based on formalized requirements. Portions of the GPS CR were modeled using the language of SRI's Prototype Verification System (PVS). During the formal methods-based analysis, numerous requirements issues were discovered and submitted as official issues through the normal requirements inspection process. Shuttle analysts felt that many of these issues were uncovered earlier than would have occurred with conventional methods. We present a summary of these encouraging results and conclusions we have drawn from the pilot project.

14. SUBJECT TERMS

Formal methods, Requirements analysis, Space Shuttle, Flight software, Global Positioning System (GPS)

15. NUMBER OF PAGES

92

16. PRICE CODE

A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		