The Pennsylvania State University

The Graduate School

Department of Aerospace Engineering

A REUSABLE SPACECRAFT COMMAND AND TELEMETRY STORAGE FRAMEWORK

A Thesis in

Aerospace Engineering

by

Shea Peterson-Burch

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2009

The thesis of Shea D. F. Peterson-Burch was read and approved* by:

Lyle N. Long
Distinguished Professor of Aerospace Engineering
Thesis Advisor

David B. Spencer
Associate Professor of Aerospace Engineering
Director of Graduate Programs, College of Engineering

George A. Lesieutre
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

* Signatures are on file in the Graduate School.

ABSTRACT

The problem of successfully managing the ever increasing quantity of spacecraft commands and telemetry involved in today's typical spacecraft missions demands building a standardized spacecraft command and telemetry storage framework that can be readily reused from mission to mission and that will allow for a broad range of applications, users, and data flow processes without requiring costly customizations. This reusable framework requires a number of key components – a flexible storage mechanism that can be adapted to a variety of mission configurations, an easy and robust mechanism for importing and modifying data, monitoring and reporting functionality that ensures data integrity and alerts the user to problems, an easy and flexible means of generating reports and transmitting information, and a means to allow for a distributed, multi-user, multi-role environment typical of most working environments. The work described here address two of the traditional problem areas associated with the construction of such a reusable command and telemetry framework: a storage mechanism that is flexible enough to handle multiple configurations as well as ongoing changes to that configuration and secondly, an import process, both bulk and incremental, that is constructed for ease of human and software use while simultaneously enforcing data integrity. Solutions are presented and subsequently implemented in Java, XML, and MySQL.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS……………………………………………………….. *vi*

LIST OF CODES…………………………………………………………………. *viii*

LIST OF FIGURES………………………………………………………………. *ix*

LIST OF TABLES………………………………………………………………… *x*

Chapter 1 Introduction……………………………………………………………. 1

1.1 Background……………………………………………………………. 1

1.2 Survey of Existing Command & Telemetry Systems…………………. 3

1.2.1 Custom SCATSF……………………………………………………. 3

1.2.2 In-house Reusable SCATSF………………………………………… 8

1.2.3 COTS SCATSF…………………………………………………….. 15

1.3 Research Motivation…………………………………………………... 18

1.4 Objectives……………………………………………………………… 21

Chapter 2 A Flexible Storage Framework………………………………………… 23

2.1 Requirements………………………………………………………….. 23

2.2 Solution Method………………………………………………………. 24

Chapter 3 File-Based Data Input………………………………………………….. 36

3.1 Overview……………………………………………………………… 36

3.2 Standardized Import File Formats……………………………………... 39

3.3 Parsing the Data……………………………………………………….. 47

3.4 Data Quality Control…………………………………………………... 48

3.5 Full and Incremental Updates………………………………………….. 52

Chapter 4 Multi-User Environment…………………………………………….. 56

4.1 Overview……………………………………………………………. 56

4.2 Implementation……………………………………………………… 56

Chapter 5 Conclusion……………………………………………………………... 60

5.1 Completed Work…………………………………….................... 61

5.2 Future Work……………………………………….……………... 61

References……..……………………………………………………………….. 63

Appendix A: Sample Data Files…………………………………………………… 67

Appendix B: Framework Software Classes………………………………………... 76

LIST OF ABBREVIATIONS

AMMOS	Advanced Multi-Mission Operations System
API	Application Programming Interface
ASIST	Advanced Spacecraft Integration and System Test
BATC	Ball Aerospace & Technologies Corp.
CCSDS	Consultive Committee for Space Data Systems
CMD	Command
COMS	Communications, Ocean, and Meteorological Satellite
COTS	Commercial Off-The-Shelf
CTDB	Command and Telemetry Database
CVT	Current Value Table
EADS	European Aeronautic Defense and Space Company
FSW	Flight Software
GEO	Geostationary Earth Orbit
GDS	Ground Data System
GSFC	Goddard Space Flight Center
ITOS	Integrated Test and Operations System
ITS	Imaging Target Sensor
JPL	Jet Propulsion Laboratory
LABVIEW	Laboratory Virtual Instrumentation Engineering Workbench
LASP	Laboratory for Atmospheric and Space Physics
LEO	Low Earth Orbit
MAESTRO	Mission Adaptive Environment for Spacecraft Test and Real-time

	Operations
MATLAB	Matrix Laboratory
MMSC	Multi-Mission Satellite Control
MRI	Medium Resolution Imager
HRI	High Resolution Imager
NASA	National Aeronautics and Space Administration
OASIS-CC	Operations and Science Instrument Support – Command and Control
OCO	Orbiting Carbon Observatory
OMG	Object Management Group
OSC	Orbital Sciences Corporation
ROS	Real-time Operations Subsystem
SC	Spacecraft
SCATSF	Spacecraft Command And Telemetry Storage Framework
SGCS	Satellite Ground Control System
SC SIM	Spacecraft Simulator
SMEX	Small Explorer Program
SQL	Structured Query Language
STOL	System Test and Operations Language
SW	Software
TLM	Telemetry
T-SQL	Transactional Structured Query Language
XML	Extensible Markup Language
XTCE	XML Telemetric & Command Exchange

LIST OF CODES

Code Listing 2-1: Hierarchy Configuration File Template………………………... 26

Code Listing 2-2: Expanded HRI Hierarchy Configuration File………………….. 28

Code Listing 2-3: Customized Hierarchy Object Data……………………………. 29

Code Listing 2-4: Relevant HRI Hierarchy Configuration File…………………… 30

Code Listing 3-1: Command Template in Plain Text Format……………………... 39

Code Listing 3-2: Example Commands in Plain Text Format…………………….. 40

Code Listing 3-3: Telemetry Template in Plain Text Format……………………... 41

Code Listing 3-4: Example Telemetry in Plain Text Format……………………… 41

Code Listing 3-5: Command Template in XML Format………………………….. 42

Code Listing 3-6: Example Commands in XML Format…………………………. 43

Code Listing 3-7: Telemetry Template in XML Format………………………….. 44

Code Listing 3-8: Example Telemetry in XML Format………………………….... 44

LIST OF FIGURES

Figure 2-1: A Simple Hierarchy………………………………………………… 25

Figure 2-2: Deep Impact HRI Imager Hierarchy………………………………….. 25

Figure 2-3: HRI Hierarchy with Filter Subsystems……………………………….. 26

Figure 2-4: Deep Impact Hierarchy……………………………………………….. 28

Figure 2-5: HRI Hierarchy Represented as a Nested Model……………………… 31

Figure 2-6: HRI Nested Model with Left and Right Values………………………. 32

Figure 2-7: Object Hierarchy Import Decision Flow Chart……………………….. 34

Figure 3-1: Example Command Data in Excel Spreadsheet Format………………. 45

Figure 3-2: Example Telemetry Data in Excel Spreadsheet Format………………. 46

Figure 3-3: Command and Telemetry Import Decision Flow Chart………………. 53

LIST OF TABLES

Table 2-1: t_object Table with HRI Nested Model Data………………………….... 32

Table 2-2: t_object_attribute Table with Custom Additional Object Information… 33

Table 4-1: t_role Table with Data……………………………………………….. 57

Table 4-2: t_user Table with Example Data……………………………………….. 58

Table 4-3: t_user_mission_role_lookup Table with Example Data……………….. 58

Chapter 1

Introduction

1.1 Background

Two critical components of most spacecraft missions include the commands used to control the spacecraft hardware and software and the science and telemetry data used to achieve the mission goals while monitoring the spacecraft health and status [20]. Integral to both of these elements is the spacecraft command and telemetry storage framework, which includes all of the databases and associated software that are commonly used to track, manage, and utilize the defined commands and telemetry over the mission life-cycle. Typically, the framework components are divided along lines of functional responsibility which include command and telemetry definition, storage, modification, reference, and mission artifact or report generation. "Commands" include all of the specific commands that are used to control the spacecraft or onboard instrument functionality, whether actively commanded from the ground or test equipment systems, or stored and used onboard the spacecraft in the flight software during autonomous or semi-autonomous control. "Telemetry" includes all of the expected spacecraft data points or engineering data that are used to monitor spacecraft status, again, for use by both onboard flight software and by ground and test systems. Additionally, the framework telemetry definitions can encompass the information being returned from any payload or science instruments [25]. Once defined, framework database components are responsible for persisting the data over the mission life-cycle while still accepting new definitions and modifications while simultaneously enforcing and maintaining data integrity. The data within such a storage framework is typically used to generate a variety of important mission data artifacts or reports. These artifacts typically include the means used to properly configure the data streams to be radiated to the spacecraft and to decommutate any data that is received from the spacecraft by the ground system equipment [30]. The data is also frequently used to configure onboard flight software intercommunication [6], as well as performing a similar function for test equipment and software [19]. Frequently, the data is mined by other mission software components such as control planning and scheduling software [12], modeling packages, mission operations displays [21], validation and verification software, and is often used to create a handbook of commands and telemetry or other ICDs for mission operators and engineers to reference.

In general, these command and telemetry storage framework software systems have many different users and uses throughout the mission life cycle. It is not atypical to have aerospace, mechanical, and electrical engineers defining hardware-level commands and telemetry, software engineers defining flight software commands and telemetry, test engineers preparing test bench data, ground system personnel working on data specific to the various ground system equipment, mission control operators revising their view of the data, and system and project engineers generally directing, adding and revising as needed. One further complication is the fact that all of the data is evolving as the mission development progresses up to, and even after, launch. The spacecraft command and telemetry storage framework acts as the central authority for managing the requirements, data, and products for of all of these users.

1.2 Survey of Existing Command & Telemetry Systems

Over the past decade, various academic, commercial, and government groups have taken a number of approaches to implementing a command and telemetry storage framework solution that meets the needs of their particular spacecraft mission or missions. Implementations tend to fall into three categories: single mission custom solutions, reusable in-house frameworks, or commercial off-the-shelf (COTS) products. The following section presents a few examples from each category in order to better acquaint the reader with specific implementations of command and telemetry framework components, how they have been used in conjunction with actual spacecraft missions, and what limitations are posed by each implementation.

1.2.1 Custom SCATSF

Custom spacecraft command and telemetry storage frameworks (SCATSF) are historically the most common implementation, especially for scientific spacecraft missions. The framework components are developed to meet the specific requirements of one spacecraft mission and are not generally designed or implemented with mission re-use in mind. Because they are targeted at only one spacecraft mission, custom SCATSFs tend to require less work in order to functionally evolve in-step with their mission; however, they do not lend themselves to being easily adapted for use by subsequent missions [31]. Besides having to start from scratch with each new mission, another downside of these custom implementations is that during flight operations, any multi-mission ground support operators need specific training to familiarize themselves with the details of the framework in order to operate the spacecraft. Three of the more recently implemented custom SCATSFs include Odin [26], Dawn-GDS [3], [4], and COMS-ROS [1], each of which exemplified these recurrent benefits and drawbacks of custom SCATSFs: namely, the creation of excellent systems that, unfortunately, cannot be easily reused [31].

Odin Space Observatory – Script-based Software System

The Swedish Odin [26] satellite is a high performance three-axis stabilized scientific spacecraft (launched in 2001) that combines earth sensing with space observations. The spacecraft was developed by the Swedish Space Corporation for astronomers and atmospheric researchers for the Swedish National Space Board and the space agencies of Canada, Finland, and France. Odin's ground station and mission operation support software is implemented as a series of custom MATLAB[1] and LabView[2] scripts that handle pre-processing, monitoring, and post-processing of the mission data. Many of the scripts rely on their interaction with definition databases of mission commands and telemetry in order to understand how to extract, present, and interpret their data [26].

The software tool chain begins with the Make Astro and Make Aero scripts. These are responsible for generating the satellite command files used for scheduling the

[1] MATLAB: A numerical computing environment and programming language developed by MathWorks

[2] LabView: A data acquisition, instrument control, and industrial automation platform and development environment developed by National Instruments

astronomy and aeronomy observations, respectively. These scripts reference Odin's database of available commands for the spacecraft and instruments. The next major script is ScheVa [26], which is used to schedule and validate the generated command files and subsequently generate a chronological list of commands to be uploaded to the spacecraft. ScheVa includes a number of sub-scripts, such as PreProcessor and Eclipse Planner, that handle tracking the spacecraft attitude and antenna selection, eclipse management, etc. via pre-planning and telemetry cross-referencing. In terms of telemetry, the post-processing script Soda is responsible for producing a mission data artifact that represents an accurate reconstruction of the attitude trajectory. Another script, OdinStatus [26], monitors the successful completion of commanded observations. To monitor spacecraft and instrument health and status, the script SHocK View uses the telemetry definition database in order to extract and interpret the data set from the downloaded and stored engineering telemetry stream. Various other scripts utilize the command and telemetry definition databases and the products of previously run scripts to handle tasks such as presenting a 3D model of the spacecraft and its line-of-sight for visualization purposes, plotting the command schedule, and evaluating the attitude control system performance.

Dawn-GDS: Dawn Ground Data System

Dawn [3], [4] is a NASA Discovery Program mission (launched in 2007) that was developed to travel to and study the largest main belt asteroids, Vesta and Ceres. The spacecraft was designed and built by Orbital Sciences Corporation (OSC) and mission operations are conducted by JPL. Dawn draws heavily on its heritage from OSC's

Orbview, Star-2, and LEOStar-2 series, although it is OSC's first interplanetary mission. OSC provided a spacecraft simulator environment (SC SIM) and used its own ground system software for integration and test activities but flight operations are conducted through Jet Propulsion Laboratory (JPL) via the AMMOS[3] ground system.

In order to support command and telemetry activities via both ground systems and both SC SIM and flight articles, the OCS Flight Software Team worked closely with the JPL's Ground Data System team in order to develop an interface between the two ground systems as well as an interface between AMMOS and the OSC flight software running in the SC SIM and on the flight articles. This functionality was broken into three components: the Dawn Command and Telemetry Database Translation toolkit, the Dawn Command Generation toolkit, and the Dawn Test Controller Assistant tool (TCAssist). The Command and Telemetry Translation toolkit is responsible for automating the translation of the existing OSC command and telemetry databases into an AMMOS format equivalent which is then be integrated into the overall Dawn GDS. The Command Generation toolkit generates SC SIM compatible commands from the AMMOS databases and is tasked with communicating these commands successfully to the SC SIM. The TCAssist component was designed to assist the OSC flight team in utilizing the Command Generation toolkit to dynamically generate Dawn commands and serves as means to expose the OSC flight team to AMMOS concepts while guiding its use [3], [4].

[3] AMMOS: Advanced Multi-Mission Operations System – a set of ground system tools developed by JPL to assist in their operation of spacecraft

COMS-ROS: Communications, Ocean, and Meteorological Satellite – Real-time Operations Subsystem

COMS [1] is a multi-mission Korean GEO satellite being jointly developed by KARI, ETRI, KORDI, KMA[4] and built by EADS Astrium for a 2009 launch. This satellite uses a system called the Satellite Ground Control System (SGCS), developed by ETRI, for satellite operations. The SGCS is divided into five subsystems: the Telemetry, Tracking, and Command subsystem (TTC), the Real-time Operations Subsystem (ROS), the Mission Planning Subsystem (MOS), the Flight Dynamics Subsystem (FDS), and the COMS Simulator Subsystem (CSS). These subsystems handle reception and processing of telemetry data, planning and transmission of commands, tracking and ranging, control and monitoring of the associated ground system equipment, analysis and simulation of the satellite, processing and analysis of flight dynamics data, and mission scheduling and reporting [8].

The ROS [1] encompasses the core of the spacecraft mission's command and telemetry system as it provides real-time monitoring of the spacecraft by ingesting telemetry it receives from the TTC subsystem, and controlling the spacecraft via sending commands through the TTC. The ROS software is dividing into six functional components: Telemetry Processing, Telemetry Analysis, Command Planning, Telecommand Processing, Data Management, and System Management. The Telemetry

[4] KARI: Korea Aerospace Research Initiative; ETRI: Electronics and Telecommunications Research Institute; KORDI: Korea Ocean Research and Development Institute; KMA: Korea Meteorological Association

Processing component receives the telemetry from the TTC, CSS, or MMSC[5] and processes it in real-time, storing for future use, and simultaneously populates various displays and triggers alarms. The Telemetry Analysis component processes the stored telemetry to provide additional trending analysis from past to future predictions. The Command Planning component translates mission timeline events into commands and provides a command sequence editor and functionality to convert command sequences into command plans. The Telecommand Processing component converts the command plans into telecommand procedures and groups the data into CCSDS standard frames for transmission to the spacecraft via the TTC, and is also responsible for verifying the spacecraft telecommand link and command execution [23]. The Data Management component manages all definitions and modification for command data, telemetry data, satellite characteristics, and system configuration data. This component is responsible for the overall database and data translation reports for use by the other ROS components and SGCS subsystems as a group. The System Management component is responsible for administrative control of the ROS subsystem – startup, shutdown, user and process management, etc. [1], [8].

1.2.2 In-house Reusable SCATSF

In-house spacecraft command and telemetry storage frameworks are developed by a particular academic, commercial, or governmental group with the guiding principle of supporting reuse over multiple missions within that group. The framework components typically have their genesis in lessons learned from a previous spacecraft mission

[5] MMSC: Multi-Mission Satellite Control, Astrium's mission control in Toulouse, France

experience. The core components may still include tools from those earlier missions, software components and processes that have been re-tasked, extended, and enhanced to support multi-mission capability. Because these frameworks are built and maintain in-house, they tend to allow for tight integration with other proprietary tools used within the same group. They also tend to readily support or adapt to in-house mission development processes and practices. However, when a spacecraft mission involves more than one such group, either as subcontractors or with responsibility for different areas of mission functionality, the inter-group communication and data translation needs can itself require additional significant time, cost, and manpower investments [27], [29].

BATC-CTDB: Ball Aerospace & Technologies Corp. - Command and Telemetry Database

The CTDB software system was initially developed for use on the NASA Deep Impact comet mission (launched 2005) and subsequently enhanced for reuse with the DARPA Orbital Express robotic satellite servicing mission (launched 2007) and further developed for the NASA Kepler planet finding spacecraft mission (launched 2009) [13]. This system manages the definition and modification of all spacecraft and instrument commands and telemetry, as well as their grouping into standard CCSDS[6] transfer frame and packet definitions, translates and generates data loading scripts for the spacecraft

[6] CCSDS: Consultive Committee for Space Data Systems – an international space data handling standards body. See http://www.ccsds.org [23].

flight and test software, as well as for the OASIS-CC[7] and AMMOS ground systems, and produces mission control operational handbooks for each of these missions [13]. BATC's CTDB's origin was a direct result of scaling issues with the company's previously script-based command and telemetry storage framework components that would be extensively customized for each new mission. The Deep Impact requirements outlining the volume of data, both commands and telemetry, marked a departure from the company's former satellite and scientific instrument experience. Taking advantage of the required "fresh start", the company also specifically tasked the developers with designing and implementing multi-mission reusability.

The CTDB software is divided into two functional tiers. The back-end data tier consists of a centralized relational database built on Microsoft's SQL Server[8] which is manipulated and monitored by a large suite of extensive stored procedures written in T-SQL[9]. The front-end client tier encompasses a custom visual basic GUI client that

---

[7] OASIS-CC: Operations and Science Instrument Support – Command and Control, a ground system developed by the Laboratory for Atmospheric and Space Physics (LASP) at the University of Colorado at Boulder and which is used internally by BATC for testing and mission operations

[8] Microsoft SQL Server is a relational database management system produced by Microsoft. Early evolutions of BATC's CTDB used a scaled down version of SQL Server called MSDE – Microsoft SQL Server Desktop Edition.

[9] T-SQL: Transactional Structured Query Language is Microsoft's proprietary extension to SQL that is enhanced with control-of-flow language, local variables, and expanded support functions.

handles data input, browsing, monitoring, and report generation. All client instances and the database tier are limited to running within the same local network for concurrent real-time access. Functionality updates to the back-end are transparent to all users, but any client updates require a coordinated re-distribution [13].

Goddard Space Flight Center - ITOS & ASIST

The GSFC has supported a number of missions over the past several years with two in-house command and telemetry systems - the Integrated Test and Operations System (ITOS) and the Advanced Spacecraft Integration and System Test (ASIST) [27]. Both systems were developed to support GSFC's approach to FSW test and spacecraft integration and test. Either system allows for swift development cycles for in-house missions as they are also pre-built to support other existing GSFC software and processes.

ITOS was originally developed to support development, integration, and testing for NASA's Small Explorer Program (SMEX). ITOS has been used on several successful SMEX spacecraft such as SAMPEX, FAST, SWAS, TRACE, WIRE, and RHESSI. More recently, it has been used on Swift, the Gamma Ray Large Area Space Telescope (GLAST), and the Lunar Reconnaissance Orbiter (LRO). A commercial version of ITOS has been made by Hammers Company. ITOS software runs on UNIX systems including Linux, Solaris, and Mac OS X and comprises an integrated set of tools for generating commands, processing, analyzing, and displaying telemetry, and performing automated operations and testing. The central component of ITOS is its command and telemetry database. The database defines all command structures and how to translate them into binary telecommand streams for uploading to the mission spacecraft. The database also defines all telemetry data including alarm limits, unit conversions, state values, and in general, how to extract or decommutate each data point from a spacecraft's telemetry stream [30]. ITOS supports numerous CCSDS data transfer formats as well as Time-Division Multiplexed[10] (TDM) data. In addition, ITOS provides extensive support for integration and test operations with the spacecraft and its subcomponents via operating over a wide range of supported data transport protocols and providing a System Test and Operations Language (STOL) interpreter [27].

ASIST was developed to support the Rossi X-ray Timing Explorer (RXTE) and the Tropical Rainfall Measuring Mission (TRMM). Specific emphasis was taken to develop an effective user interface as part of human factor studies done by the Flight Telerobotic Servicer lab. ASIST's success led to its further use on the EO-1, WMAP, IMAGE, ST-5, and SDO missions. From an aerospace software perspective, "ASIST is a scalable real-time command and control system for spacecraft development, integration and test, and operations."[11] ASIST's software is architected such that it allows parallel commanding and distributed telemetry decommutation from a variable number of low cost COTS computers running Linux. A master workstation controls the overall network. At its core, ASIST is built around a command database, a separate telemetry database, and a number of stored procedures that operate over both configuration and definition

[10] TDM, or Time Division Multiplexed data refers to the process of sending multiple bit streams simultaneously over a channel by dividing the streams into sub-channels that each take fixed time-slots.

[11] Pfarr et al, "Proven and Robust Ground Support Systems," p. 7.

datasets. A built-in STOL interpreter and a centralized Current Value Table (CVT) that tracks the latest telemetry point values together allow for script and procedure based access to both command and telemetry definitions for integration, testing, and operations [27].

OSC – MAESTRO: Orbital Sciences Corporation - Mission Adaptive Environment for Spacecraft Test and Real-time Operations

MAESTRO, the Mission Adaptive Environment for Spacecraft Test and Real-time Operations [29], was developed by Orbital Sciences Corporation to provide a more cost-effective solution for ground software versus similar functionality provided by COTS products that the company found to be too cost-prohibitive for the types of LEO and GEO satellite missions the company was pursuing. MAESTRO has been used on a number of spacecraft missions since its first use on the SeaStar mission. These missions include MubLcomm, DART, GALEX, ACRIMSAT, OV-3, OV-4, Telkom, QuikTOMS, and recently, on the Interstellar Boundary Explorer (IBEX) [5], launched 2008, and the development of NASA's Orbiting Carbon Observatory (OCO) that tragically failed to achieve fairing separation upon its launch in 2009 [33], and on the upcoming NASA Glory mission launching in late 2009 [29], [34].

The MAESTRO software framework includes functionality to support command and telemetry handling, level zero processing,[12] pass planning,[13] and event notifications.

[12] Level 0 generally refers to commands and telemetry that apply to a spacecraft's safe-mode, i.e. a lower or more basic level of data and functionality that allows the spacecraft and operators to recover from an event that leaves the spacecraft in an off-nominal state.

All command and telemetry definitions, processing rules, packet groupings, conversion functions, constraints, and enumeration are generated by the Composer component and stored in the command and telemetry database. In addition to the command and telemetry database itself, two other key components of MAESTRO are the DataServer, which handles all telemetry stream processing, and the Commander, which formats and sends the commands to the spacecraft. Each relies heavily on its interface to the command and telemetry database in order to generate the rules and actions to perform their functional tasks. The Commander supports real-time commanding and absolutely or relatively timed sequences of commands. Sequenced commands can be grouped into command blocks to enhance uplink bandwidth use and to facilitate pre-planned pass scripting which is enhanced by the inclusion of a STOL interpreter. Another key MAESTRO component is MAIDS which provides an automated monitoring and notification system. MAIDS operates by automatically performing actions based on pre-defined criteria. These criteria can be based on telemetry values, system states, or time values. Actions can include sending emails, creating backups, and kicking-off other command script. The Archiver interfaces with the other MAESTRO components to store each of their activities as well as being responsible for archiving all telemetry received from the spacecraft [29].

[13] A pass refers to the mission operational period when a spacecraft is in communication with a particular ground station – i.e. its current orbital location is "passing" over the transceiver communications equipment.

1.2.3 COTS SCATSF

Commercial off-the-shelf (COTS) frameworks usually provide a turn-key solution for all or a targeted subset of mission developers' needs in terms of managing their spacecraft commands and telemetry, whether for development, integration, test, or operations. Any modifications to the COTS product code-base to support enhancements, bug fixes, and maintenance can usually be handled by the COTS vendor through additional purchases, upgrades, or engineering services contracts [2]. There are several pros and cons to using COTS solutions [35], with most cons being solvable through sufficient ongoing monetary investments. Because most spacecraft missions have at least some unique characteristics, either some modification to a COTS one-size-fits-all product is required or alternatively, compromises with mission requirements will need to be managed. Custom glue code is usually required when supporting a non-comprehensive COTS solution within the context of the rest of the spacecraft mission development, integration, test and operations – usually in the communication or messaging layer. Additionally, an established group can become tightly coupled to, and dependent upon, the continuing to use the COTS solution, making any future transition to another SCATSF difficult [27]. Timely support is usually an expensive proposition; however, the vendor is usually highly motivated to keep their paying customers satisfied. Also on the plus-side, the SCATSF functionality will have a proven track-record, a history of incremental improvements, small initial development timelines, and can be used when the necessary experience is not available in-house.

L-3 InControl-NG: L-3 Communications InControl-NextGeneration

InControl-NextGeneration [11] is a turn-key COTS SCATSF solution originally developed, sold, and supported by the Storm Control Systems[14] subsidiary of L-3 Communications. InControl-NG supports a wide range of command and control system requirements including telemetry processing, spacecraft and payload commanding, data display and analysis, spacecraft fleet monitoring and control, onboard system management, and ground equipment monitoring and control. Multi-satellite support is built-in; including heterogeneous spacecraft fleets from various manufacturers. This system supports satellite development, test, and on-orbit operations. Key features of the software include straightforward database integration, rapid development and validation of test sequences, automated report generation, dynamic limit generation, pre-test validation of test sequences and monitor and control of multiple test suites with diverse interfaces. Automated operations, external interface support through exposed APIs, archiving, archive retrieval, and data analysis are also integrated within the functionality of the InControl-NG software suite [10]. InControl-NG has been used on a number of commercial and military spacecraft including EADS Astrium's Skynet 4, Skynet 5, the U.S. Air Force Advanced Extremely High Frequency (AEHF) military satellite [11].

Harris Corporation - OS/COMET

OS/COMET, from the Harris Corporation, has been used on a wide variety of commercial and governmental missions such as the Iridium LEO satellites, the GPS MEO satellites, and was selected in 2008 to be incorporated into NASA's Constellation

[14] Storm Control Systems was integrated into L-3 Telemetry West (L-3 TW) in 2005

Launch Control System [32]. OC/COMET was designed with an emphasis on being adaptive to the various business model needs of its customers while still meeting all technical requirements. The software is comprised of a suite of eight core components, or “Facilities”, each responsible for a specific satellite control task. An additional set of seven software tools is available in an add-on basis and are used to extend the core OC/COMET functionality. All components and tools communicate over the OS/COMET Software Bus [9].

The System Administration Facility tracks and manages all OS/COMET resources and processes. The Communication Services Facility provides and manages inter-process messaging. The Symbol Processing Facility manages OS/COMET symbols which are representations of specific data items along with their associated attributes such as unit conversion, alarms, process status, etc. Symbols apply to both commands and telemetry and are organized into collections of symbols called MFiles. The Data Distribution Facility monitors the status of all MFiles and is responsible for distributing updates on an MFile to any other process that is using it. The Telemetry Processing Facility handles decommutating any received telemetry stream data into individual symbols. The Command Processing Facility uses a combination of OS/COMET software components to generate and transmit commands to a target spacecraft or payload. The Command Processing Facility relies on command definition files, specified for each target, which are used to generate a run-time command database for that target. The Language Processing Facility is responsible for parsing text input in a large variety of OC/COMET and 3rd party formats. The final core component is the Recording and

Logging Facility, which provides a means of recording or archiving raw and decommutated telemetry as well as any other OC/COMET system events [9].

The OA/Tool provides an integrated timeline for coordinating control center activities and also performs operation automation tasks. The GC/Tool provides a monitoring and control tool for use with ground equipment. The FA/Tool is a rule-based fault detection and resolution tool for supporting fault automation. The AR/Tool works with the archived data by providing retrieval and management functionality. The SIM/Tool is a simulator tool used for simulating control center data during development and test. The MEM/Tool allows mission operators to interact with and manage on-board computer memory. The XTCE Translator provides translation services for porting data between XTCE[15] and OS/COMET source files [9].

1.3 Research Motivation

With the increasing adoption of higher level programming languages, such as C/C++ and Java [16], [18] for meeting the varied requirements of spacecraft missions, and the increasingly sophisticated on-board and ground-based environments available [19], space programs have been moving inexorably from hardware to software-based spacecraft control and monitoring. With the rise in functionality such changes afford, such as autonomous control and precise status monitoring, has come a corresponding rise in data complexity. As these changes have gained momentum across the industry, the problems posed by a lack of a standardized and reusable command and telemetry

[15] XTCE: XML Telemetric & Command Exchange is a standard information model for exchanging command and telemetry data between groups [7].

framework have become more manifest and more difficult to overcome. The increase in the sheer amount of data needing to be processed for each spacecraft mission requires an alteration in the ways aerospace companies organize and track spacecraft commands and telemetry [21], [24]. At Ball Aerospace and Technologies Corp., for example, former missions typically involved hundreds to a few thousand commands and telemetry while newer missions by the company, such as Deep Impact, now involve tracking hundreds of thousands of data points. Additionally, as multiple groups are typically involved with different stages or components in the spacecraft development life-cycle, functional areas inevitably need to interact with the both the underlying command and telemetry data and each other in meaningful ways that are automatic and/or process driven.

To handle this information and functional interaction overload, many groups typically develop a system from scratch; designing, programming, and implementing a new, custom command and telemetry system for each and every mission. In the days in which spacecraft missions were more limited in terms of communication bandwidth and in general more hardware control oriented, such an onus was not perhaps entirely overwhelming. But in today's market, the need to "reinvent the wheel" for each and every mission has become unacceptable in terms of time, manpower, cost, and reliability. The first step in improving this organization, of streamlining some of the processes by which the aerospace industry functions as a whole, is the construction of a basic, robust, reusable command and telemetry storage framework that can be easily adapted from mission to mission. Many aerospace companies have recognized this need and begun a series of in-house developments to create just such a reusable framework [12], [14], [19], [27]. Unfortunately for these purposes however, many of the larger spacecraft missions to date—Deep Impact, Orbital Express, James Webb Space Telescope [28], etc.—are the products of multi-company and government group collaborations. Even if each company involved in such a project develops its own reusable interface, the discreet elements of the mission as designed by the individual companies will generally still be functioning under different mandates, with different standards that require further custom integration and translation work as the mission progresses.[16]

In this body of work, I want to identify and address certain difficulties I had encountered with such storage frameworks while working with Ball Aerospace and Technologies Corp., JPL, Boeing, and DARPA on the Deep Impact, Orbital Express, and Kepler spacecraft missions. What I propose is the construction of the underpinnings of a basic, open-source, command and telemetry storage framework that will provide two basic functions. The first is to provide an initial, solid SCATSF foundation for any particular group to further customize, and thus save individual companies wasted time and manpower for constructing completely custom systems for each new mission while also sparing them from going through an iterative lessons learned development processes. Secondly, to provide a built-in way to simplify and standardize the flow of command and telemetry information within aspects of the aerospace industry itself, allowing for a smoother collaborative process between groups while reducing the possibilities of error that can arise from using competing and conglomerate systems. Such benefits could be particularly felt in the long-term Mars projects currently in development across the industry [20].

[16] Note that the creation of robust, reusable software will also increase a given company's abilities to reuse hardware elements, such as satellite transceivers [22].

1.4 Objectives

I envision a web-based interface with secure, global distributed services access that guides both the input and modification of data, provides the means to review the mission spacecraft's command and telemetry, and implements a flexible data reporting mechanism for generating various data artifacts for other mission software components. A practical reusable command and telemetry system requires the following core components:

1. A flexible storage mechanism that can handle multiple spacecraft configurations,
2. An easy and robust mechanism for defining, importing, and modifying data,
3. Monitoring and reporting functionality that ensures data integrity and alerts the users to any problems,
4. An easy and flexible means of generating reports, translating datasets, and transmitting information,
5. A means to allow for a multi-user, multi-role environment typical of most working environments.

The specific purpose of this project is to address two of the traditional problem areas associated with the construction of such a reusable command and telemetry framework: a storage mechanism that is flexible enough to handle multiple configurations as well as ongoing changes to that configuration and secondly, an import process, both bulk and incremental, that is constructed for ease of human and software use while simultaneously enforcing data integrity. Chapter 2 outlines and demonstrates the construction of such a framework using entity-relationship diagrams for relational database systems, which are then implemented in XML, Java, and MySQL. Chapters 3 concerns the construction of the data input process, specifically bulk data import. This involves the creation and parsing of standardized file based inputs in multiple formats; addressing issues of data quality control- specifically the means of ensuring consistent standardization and validation of incoming data is discussed; and also outlines the means of allowing for both full and incremental updates using both individual components and entire data sets. Chapter 4 then describes the access control and assignation of roles necessary for the implementation of data importation in a multi-user environment. Chapter 5 summarizes the completed research and presents a roadmap for future work involving other components of the reusable command and telemetry storage framework.

Chapter 2

A Flexible Storage Framework

2.1 Requirements

A truly reusable system must allow for a broad range of eventual uses, such as for multiple spacecraft, multiple spacecraft systems (and their subsystems ad infinitum), support for any division between spacecraft bus versus payload and instrumentation (and their subsystems ad infinitum), accommodation for hardware versus software only elements, etc – including supporting all of these simultaneously. In addition, to be truly practical in a typical business or project environment, the system needs to allow for any data flow processes unique to the mission or business. For example, while an artifact produced from the command and telemetry storage framework might be a comprehensive spacecraft operator's commanding manual or instructions (human or software) on how to properly decommutate the telemetry being received by the ground systems, the individual commands and telemetry to be stored are rarely defined all at once, nor are they static throughout the lifecycle of the mission. An increasing number of missions now make substantial command and telemetry changes even after the launch, whether to accommodate fixes or to support entirely new functionality. The ephemeral nature of software and its increasingly dominating role in spacecraft control now require many of the supporting software systems, such as the command and telemetry storage framework, to be as responsive. Additionally, command and telemetry definition responsibilities are quite often distributed among various teams both within a company as well as outside it among various subcontractors or other groups that are each responsible for their portion of the spacecraft mission. Eventually, these disparate definitions need to be integrated in a way that is both comprehensive and flexible.

2.2 Solution Method

In order to design for such a broad spectrum of requirements, I chose to implement a means by which the initial setup of the storage framework (and any subsequent changes) can be auto-configured based-off of a hierarchical model whose specific definition is determined by the using party and which could therefore encompass an entire mission or even perhaps just a single subsystem. The required model can be simply and intuitively implemented as an object hierarchy tree in a XML file. This hierarchy represents a model of the mission, whether it is configured based on hardware, software, or other form of modeling, or some combination thereof. This hierarchy creates a skeleton upon which to "hang" any commands and telemetry in a way that makes sense to the planners and users of the mission, rather than forcing those people to make use of a preconfigured command and telemetry framework that may not be appropriate for their mission. This "skeleton" is also flexible enough that future commands and telemetry can be associated with it at any or all levels, or even with entirely new levels that are first defined in an updated hierarchy model.

For example, a simple hierarchy could involve a single spacecraft and its subsystems (see Figure 2-1).

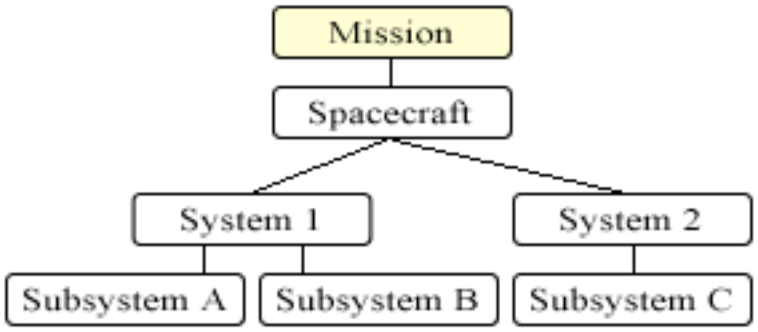


Figure 2-1: A Simple Hierarchy

The project personnel would create a mission label, a spacecraft label within that mission, and elaborate to any extent the system and subsystems within that spacecraft, all in an XML configuration file. Most missions require a great deal of elaboration, but this framework that I develop can allow for extreme complexity and for extreme simplicity, as required by any given group. For example, on the NASA Deep Impact Mission, certain personnel were concerned the process of image-capturing via the High Resolution Imager (HRI) telescope mounted on the flyby spacecraft instrument platform. These personnel, when using this framework, might begin by creating the following basic hierarchy (see Figure 2.2):

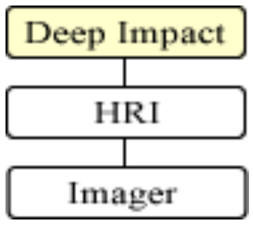


Figure 2-2: Deep Impact HRI Imager Hierarchy

This represents the simplest of mission formats: single mission, single instrument, single subsystem. Having created this hierarchy model, the HRI telescope team might add a series of subsystems to control which filter is used for image capturing at a given time (see Figure 2-3).

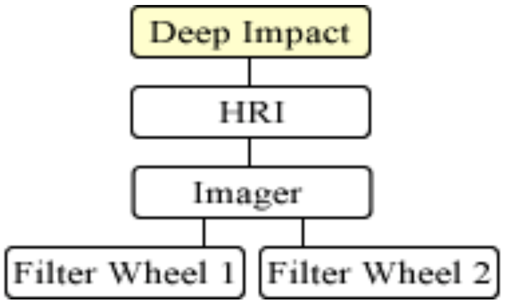


Figure 2-3: HRI Hierarchy with Filter Subsystems

The HRI team would then transfer this model into a XML configuration file[17] starting from the basic template provided:

```
<database>DATABASE NAME
   <mission>MISSION NAME
      <description>MISSION DESCRIPTION</description>
      <object>SPACECRAFT NAME
         <description>SPACECRAFT DESCRIPTION</description>
         <object>INSTRUMENT NAME
            <description>INSTRUMENT DESCRIPTION</description>
            <object>INSTRUMENT SUBSYSTEM 1 NAME
               <description>INSTRUMENT SUBSYSTEM 1 DESCRIPTION
               </description>
            </object>
            <object>INSTRUMENT SUBSYSTEM 2 NAME
               <description>SUBSYSTEM 2 DESCRIPTION</description>
            </object>
         </object>
      </object>
   </mission>
</database>
```

Code Listing 2-1: Hierarchy Configuration File Template

[17] I have chosen XML because this particular language allows for general adaptability across software, is easily readable for both humans and software, has universality of industry use, and intrinsically lends itself to specifying hierarchies. See Appendix A for detailed examples of configuration files.

Because the HRI team is working within a company that is concerned with more than a single subsystem, the team may wish to create a more fully expanded hierarchy within the XML configuration file, encompassing additional details about the mission and the individual spacecraft upon which the imager is mounted, as well as their own specific payload and subsystems. This will allow the HRI team's commands and telemetry to be readily integrated into a more comprehensive set of Deep Impact commands and telemetry at a later point.[18] In such a case, the hierarchy model might resemble the following (see Figure 2-4):

[18] Recall that the framework does not force a particular data definition strategy or even a specific timeline to define the data to be stored. Therefore rather than a number of independent instances of the command and telemetry framework encompassing sub groupings of commands and telemetry, a particular mission may choose to create a single instance of the framework and define their hierarchy all at once and subsequently allow sub groups to access and manipulate their portion of the hierarchy. To provide flexibility, these choices are left to the users of the framework to make based on what makes the most sense to them and their business needs.

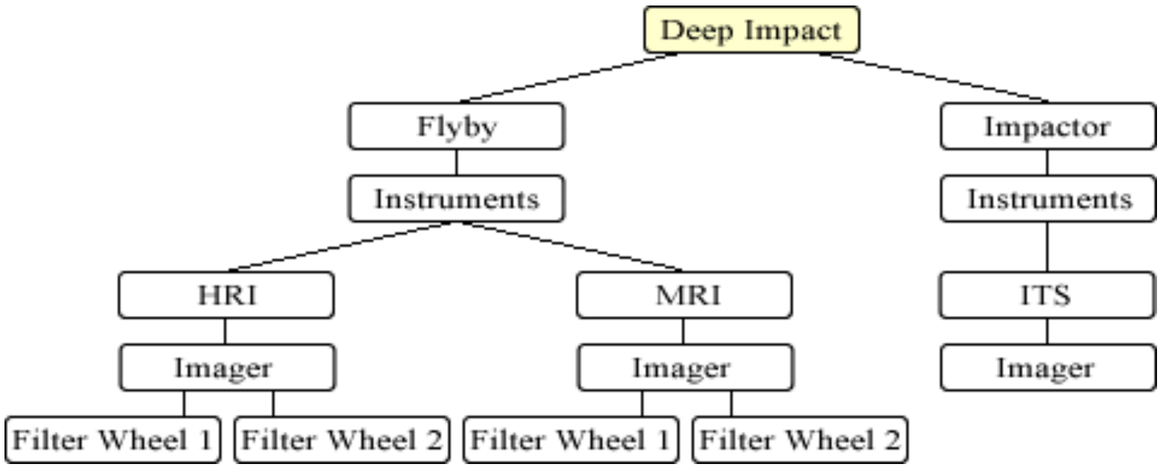


Figure 2-4: Deep Impact Hierarchy

From which the XML file to be submitted might resemble the following:

```
<database>CTDB
   <mission>Deep Impact
      <description>NASA's Deep Impact comet cratering mission -
           exploring the origins of our solar system.
      </description>
      <object>Flyby Spacecraft
         <description>The main transport spacecraft</description>
         <object>Instrumentation Platform
            <description>Mobile platform for all instruments
            </description>
            <object>HRI
               <description>High Resolution Imager Telescope
               </description>
               <object>Imager
                  <object>Filter Wheel Assembly 1
                     <description>Series 1</description>
                  </object>
                  <object>Filter Wheel Assembly 2
                     <description>Series 2</description>
                  </object>
               </object>
            </object>
            <object>MRI
               <description>Medium Resolution Imager Telescope
               </description>
               <object>Imager
                  <object>Filter Wheel Assembly 1
                     <description>Series 1</description>
                  </object>
                  <object>Filter Wheel Assembly 2
                     <description>Series 2</description>
                  </object>
               </object>
            </object>
         </object>
      </object>
      <object>Impactor Spacecraft
         <description>The comet cratering spacecraft
         </description>
         <object>Instrumentation System
            <description>System managing all onboard instruments
            </description>
            <object>ITS
               <description>Impactor Targeting Sensor
               </description>
               <object>Imager</object>
            </object>
         </object>
      </object>
   </mission>
</database>
```

Code Listing 2-2: Expanded HRI Hierarchy Configuration File

In addition, a particular group may wish to customize their various hierarchy objects with additional information, such as mnemonic devices, more detailed descriptions, or other attributes of the object. This template allows for such additions without entailing further complications and database or coding alterations.[19] The user can simply add any required information at a given level. So, for example, if the HRI team desired to add details concerning the multiple filters present on the filter wheel series 1 but they do not wish to break them out into their own hierarchy objects, they could simply encode that information as the following custom XML tags:

```
               <object>Filter Wheel 1
                  <description>Series 1</description>
                  <filter count>3</filter count>
                  <filter>Filter 1</filter>
                  <filter>Filter 2</filter>
                  <filter>Filter 3</filter>
```

[19] User-defined object attributes do allow for a convenient "hook" for any customized framework functionality that a particular group wishes to add such as mode management.

```
                </object>
```

Code Listing 2-3: Customized Hierarchy Object Data

In cases where an individual company or group has been subcontracted to deal with a single component, however, such a complex hierarchy may be unnecessary. Were the HRI team in such a situation, the hierarchy constructed for the template could be much simpler, foregoing the tiers not directly relevant to them, such as spacecraft and the instrumentation platform (refer back to Figure 2-3). In that case, the completed XML configuration file might resemble the following:

```
<database>CTDB
   <mission>Deep Impact
      <description>NASA's Deep Impact comet cratering mission -
           exploring the origins of our solar system.
      </description>
      <object>HRI
         <description>High Resolution Imager Instrument
         </description>
         <object>Filter Wheel 1
            <description>Series 1</description>
            <filter count>3</filter count>
            <filter>Filter 1</filter>
            <filter>Filter 2</filter>
            <filter>Filter 3</filter>
         </object>
         <object>Filter Wheel 2
            <description>Series 2</description>
         </object>
      </object>
   </mission>
</database>
```

Code Listing 2-4: Relevant HRI Hierarchy Configuration File

Should such a group be working in isolation, the construction of such a simplified hierarchy, foregoing several levels integral to the completed mission, would not pose a difficulty upon reintegration with the whole. The sub-hierarchy can simply be reintegrated into a comprehensive mission hierarchy configuration file at an appropriate point as determined by the project coordinator and an incremental update be made to the database as whole. (Incremental updates will be addressed in Chapter 4.) Thus, by virtue of the extreme flexibility of this template, even those groups working in isolation from the mission can have their hierarchy quickly, easily, and without emendation inserted into the more complex hierarchy of the entire mission without undermining their own data integrity and without undermining the integrity of the rest of the hierarchy. Moreover, such an insertion would not disrupt command and telemetry definitions that may have already been added to other components of the entire mission hierarchy.

Once the hierarchy configuration file has been constructed and submitted, the framework software must parse it and re-encode it into a form suitable for storage in a relational database, i.e. a table designed to store an arbitrary hierarchy. A robust means of accomplishing this is to restate the hierarchy model (refer back to Figure 2-4) as a nested set [17]. In the case of the Deep Impact HRI camera team, the nested set model would be represented by the following: (see Figure 2-5).

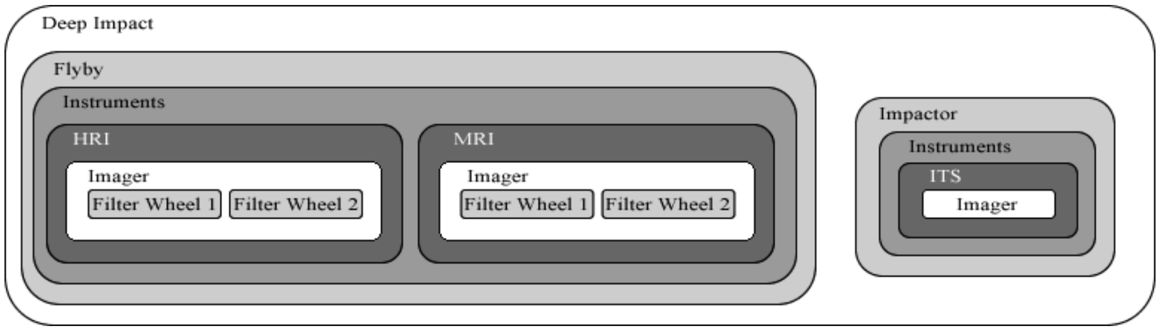


Figure 2-5: HRI Hierarchy Represented as a Nested Model

To represent how this nested set model is implemented in an actual relational database table [15], I assign numbers to the left and right of each object, moving from left to right sequentially (see figure 2-6).

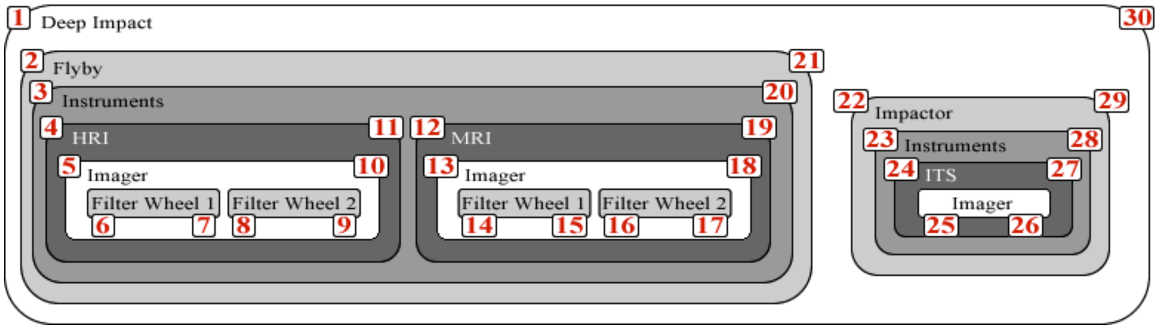


Figure 2-6: HRI Nested Model with Left and Right Values

Every record in the table, from hence "t_object", represents one hierarchy object, each with a unique record id and an associated mission id. The mission id is included so that multiple missions can be defined with a single instance of the command and telemetry storage framework if needed. Each record also contains the "left" and "right" values that define the outer perimeters the object in relationship to those that surround it or are within it.

object_id	mission_id	object_name	lft	rgt
1	1	Deep Impact	1	30
2	1	Flyby Spacecraft	2	21
3	1	Flyby Instruments	3	20
4	1	HRI	4	11
5	1	HRI Imager	5	10
6	1	HRI Filter Wheel 1	6	7
7	1	HRI Filter Wheel 2	8	9
8	1	MRI	12	19
9	1	MRI Imager	13	18
10	1	MRI Filter Wheel 1	14	15
11	1	MRI Filter Wheel 2	16	17
12	1	Impactor Spacecraft	22	29
13	1	Impactor Instruments	23	28
14	1	ITS	24	27
15	1	ITS Imager	25	26

Table 2-1: t_object Table with HRI Nested Model Data

If the user has placed additional information in the configuration file, as in the case of the HRI series 1 filters (refer back to Code Listing 2-3), additional records are placed in a separate table, from hence "t_object_attribute," that contains an association to the appropriate record's object_id in the t_object table.

object_attribute_id	object_id	attribute_name	attribute_value	order
1	6	filter count	3	1
2	6	filter	Filter 1	2
3	6	filter	Filter 2	3
4	6	filter	Filter 3	4

Table 2-2: t_object_attribute Table with Custom Additional Object Information

For these tables, SQL queries and stored procedures have been written to access or modify the hierarchy at any point or in its entirety. The framework software's import classes execute these queries and stored procedures appropriately as it process the imported configuration file. These queries and procedures include the following functionality:

- Inserting new objects
- Updating existing objects
- Deleting existing objects (and any nested objects)
- Moving an existing object to another location within the nested model

I have implemented this element of the framework software in Java code[20] such that it corresponds to the following decision flow chart (see Figure 2-7):

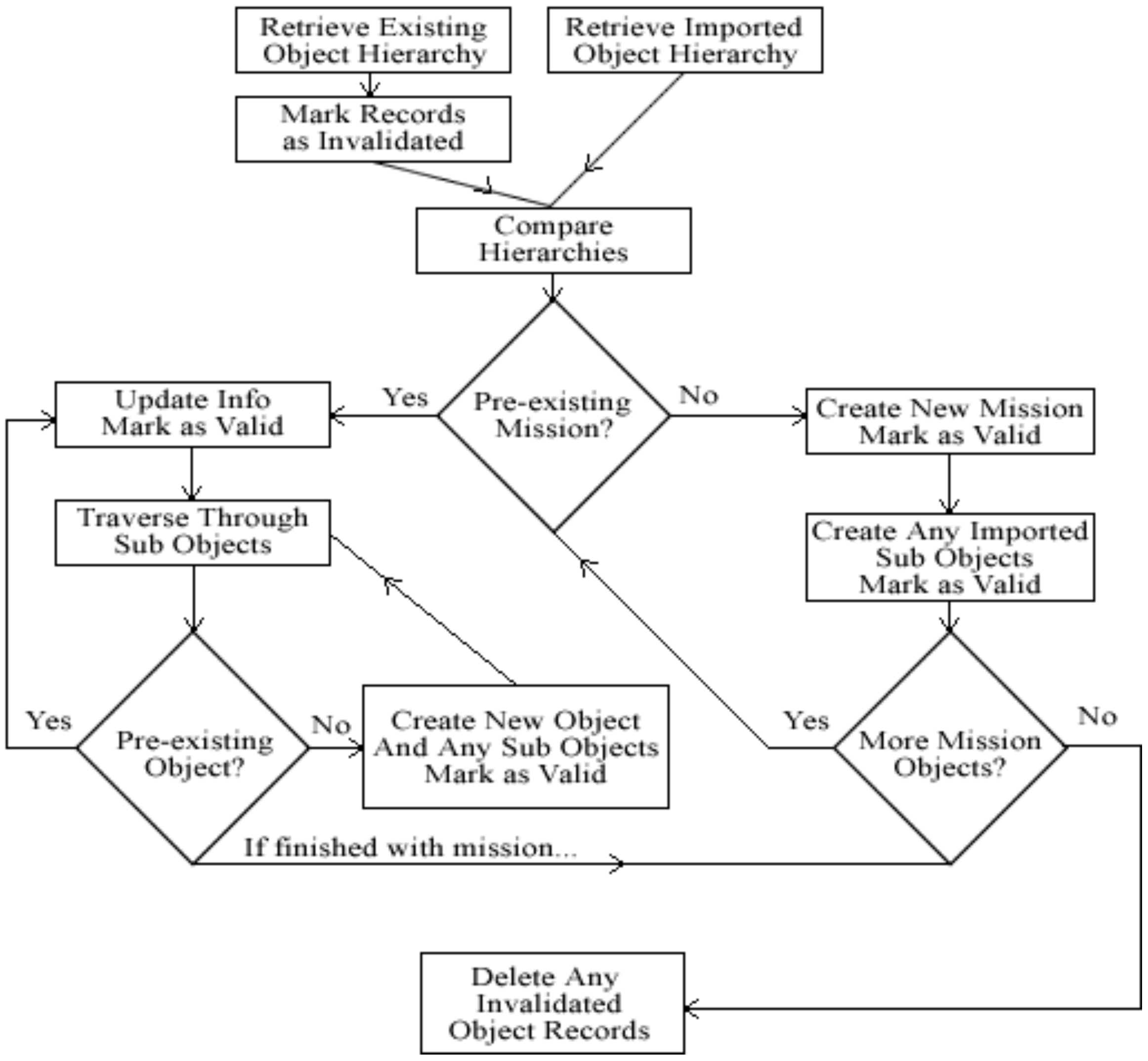

Retrieve Existing Object Hierarchy
Retrieve Imported Object Hierarchy
Mark Records as Invalidated
Compare Hierarchies
Update Info Mark as Valid
Yes
Pre-existing Mission?
No
Create New Mission Mark as Valid
Traverse Through Sub Objects
Create Any Imported Sub Objects Mark as Valid
Yes
Pre-existing Object?
No
Create New Object And Any Sub Objects Mark as Valid
Yes
More Mission Objects?
No
If finished with mission...
Delete Any Invalidated Object Records

Figure 2-7: Object Hierarchy Import Decision Flow Chart

[20] Like C/C++, the Java language already has libraries pre-built to work with XML files and databases. I have chosen to code in Java due to its native thread and memory handling and due to the large set of preexisting class libraries, but this code could be ported to C/C++ if desired.

When the process detects a move, i.e. it has found a matching pre-existing object that is located at a different point in the hierarchy, all sub-objects are also moved to the new location. Any pre-existing command and telemetry data associated with valid object records remain untouched. However, if any such data is still associated with an invalidated object record when the import process completes, it will be deleted as well.

At this point, I have constructed and implemented a highly flexible hierarchy, one that is eminently suitable to address a wide variety of industry needs, and that allows for multiple levels of usage and modification. In the next section, I will address the preliminary issues of actual command and telemetry data importation into the database as constructed through this framework.

Chapter 3

File-Based Data Input

3.1 Overview

Aside from issues with flexibility in command and telemetry construction, the other major hurdle posed to the creation of a reusable SCATS is the data importation process. In today's more complex spacecraft systems, the sheer volume of data points, and the number of persons involved in inputting those data points, requires a comprehensive, standardized, consistent approach to data entry. Data entry can generally be made in two different ways: first, through a file-based import process that reads previously prepared files and does a massive update of the database, and secondly, through direct user interaction with web pages allowing the user to enter and/or modify data directly into the system.[21] Both of these data input processes must be accounted for in the completed framework as they fulfill different requirements. A real-time, web-based interface would to a great extent eliminate the need for a separate data cleaning step due to the availability for real-time checks built directly into the inputs of prefabricated forms. This interface also provides the means to make immediate changes to the database. But while individual, human users may find such an interface quite useful for making minor data modifications, the command and telemetry framework must also include options for file-based imports, both to provide a means to quickly make large quantities of changes, and also because such a web-based interface cannot easily be utilized by other software tools. I choose to focus on file-based importation because the

[21] The real-time web interface implementation is outside the scope of this present work.

process is more fundamental, providing the means to serve the needs of both humans and software while providing the greatest range of flexibility for any number of variants in data flow processes and/or external tools.

Having previously established the mission hierarchy, the user must choose the format in which s/he intends to submit the command and telemetry data. In the case of file-based imports into the database, it is necessary to establish a series of standard formats that govern the ways in which those imported files are constructed. I have chosen two basic types of importation formats—flat files and Excel files—based on ease of use for both human beings and other software tools operating in conjunction with the SCATS. Should the data be imported via flat files, my database will allow those files as either plain text files (comma-separated, tab-separated, etc.) or XML files[22]. Plain text files will be convenient for companies that already utilized pre-existing, non-XML tools. Also, variations of plain text files are the current non-database standard for transferring information between two disparate pieces of software. XML files have the virtue of inherent organization; moreover, XML is becoming the industry standard for data transfer between components. The XML language may not be the simplest for the average user to understand, but it is logical enough to be readable and thus provide a bridge between the human user and the software. The second option for data transfer formatting is that of Microsoft Excel. This choice was made for the obvious reasons of prevalence, user familiarity, and ease of accessibility. Admitted, Excel has it flaws, but these are more than outweighed by its ease and universality of use.[23]

[22] For example code, see Appendix B.

[23] For examples of each import file, see Appendix A.

As the user begins to enter the data into the system, s/he will be faced with a series of prompted decisions (see figure 2.1 flow chart). Before inputting any data values, the user can specify whether s/he intends to perform a full data import (to completely replace the command and telemetry content) or a partial data import (to add or replace only a subsection of content). For the user's convenience, my framework can generate templates for each of these acceptable formats. The user will be prompted to specify the format—plain text, XML, or Excel—that they prefer. The framework will then generate as complete a template as possible; in the case of a full data import, the template will be based on the established hierarchy, and in the case of a partial import, the template will be based both on the relevant portion of the hierarchy and the previously entered data affected by the new import.[24] This template is not technically necessary; a flat or Excel file could be directly provided by the user. What the template provides, however, is a guarantee that the data to be entered matches the hierarchy; the template also will take into account any commands and telemetry data already defined. In the case of a partial import, the template also includes preexisting data that will be affected by the new import. From the user's perspective, however, the differences between templates generated for full and partial imports will be minimal; each template will be similar in format and appearance, the difference lying only in the amount of data that the user is prompted to enter. After specifying the format, the user can choose whether s/he wants to be prompted immediately upon the completion of the validation

[24] By "affected data," I refer to pre-existing command and telemetry data that is related to the data being altered but not a part of it, such as any telemetry data dependent upon the command data being altered.

process on a table-by-table basis or upon completion of the entire data validation process. (Data validation will be discussed further in section 3.4)

3.2 Standardized Import File Formats

The data within either the flat file or the Excel file must conform to a series of formatting regulations which will allow the parsing code to accurately read the data in question. In order to help enforce these standards, the user is first required to separate their command data from telemetry data and place it within the context of one of the available file format options. While these files are typically first generated from the database and then edited, this is not required. Users can also break their data up across multiple files if desired as the import process will read through all files that reside in a specified folder during execution. The user simply needs to ensure that command data is being imported from a separate folder than telemetry data. The general import process will be discussed in greater detail in section 3.5.

To begin, for working with plain text flat files, data columns are tab separated; the first row of data contains column titles; and each row must adhere to the standard layout presented in the following Code Listings 3-1 through 3-4. Column name have been placed in **bold**. Additionally, any field values that include spaces or commas must be quoted.

```
mission_id  mission     object_id   object_path cmd_id
      old_cmd_name      new_cmd_name      cmd_desc    cmd_type
      cmd_bit_length    delay predecessor is_critical cmd_constraints
      resultant_state   tlm_response      cmd_param_id
      param_number      old_param_name    new_param_name    param_desc
      default_value     value EU    input_type  conv_to_DN  range
      data_type   param_bit_length
```

```
"DATABASE MISSION ID (if available)"      "MISSION NAME"    "DATABASE
OBJECT ID (if available)"     "COMMA SEPARATED LIST OF PATH TO OBJECT"
      "DATABASE CMD ID (if available)"    "OLD CMD NAME (required if
updating existing data)"      "NEW CMD NAME (required for new cmds)"
      "CMD DESCRIPTION" "COMMAND TYPE"    "NUMBER OF BITS IN CMD" "ANY
DELAY (if required)"    "NOTES ABOUT ANY CMDS THAT MUST BE RUN BEFORE
THIS CMD"   "TRUE OR FALSE"   "CMD CONSTRAINT INFO"   "RESULTANT STATE
INFO" "ADD ANY NUMBER OF TLM RESPONSES"   "DATABASE CMD PARAM ID (if
available)" "ORDER OF PARAMETER WITHIN THE CMD" "OLD PARAMETER NAME
(required if updating existing data"      "NEW PARAMETER NAME (required
for new parameters)"    "PARAMETER DESCRIPTION" "DEFAULT VALUE (if
available)" "SPECIFIC VALUE (if available)"     "ENGINEERING UNITS"
      "CONVERSION TO DATA NUMBER (if needed)"   "ALLOWABLE VALUE RANGE"
      "PARAMETER DATA TYPE"   "NUMBER OF BITS IN PARAMETER"
```

Code Listing 3-1: Command Template in Plain Text Format

```
mission_id  mission     object_id   object_path cmd_id
      old_cmd_name      new_cmd_name      cmd_desc    cmd_type
      cmd_bit_length    delay predecessor is_critical resultant_state
      cmd_constraints   tlm_response      cmd_param_id
      param_number      old_param_name    new_param_name    param_desc
      default_value     value EU    input_type  conv_to_DN  range
      fsw_var_name      data_type   param_bit_length  pkt_id
      pkt_path    pkt_start_bit
      Deep Impact       "DI,Flyby,ADF,APPID18,PKTID1"
      ADFOP_INPUTS      Set orbit propagation inputs  SWC   2024
      FALSE                   1           ORBSTATEPOSX      Point
to propagate from - Position X            KM
      orbital_state_position.Vec[0] flt64 64          "appid18,pktid1"
      0

                        2           ORBSTATEPOSY      Point to
propagate from - Position Y               KM
      orbital_state_position.Vec[1] flt64 64          "appid18,pktid1"
      64

                        3           ORBSTATEPOSZ      Point to
propagate from - Position Z               KM
      orbital_state_position.Vec[2] flt64 64          "appid18,pktid1"
      128

                        4           ORBSTATEVELX      Point to
propagate from - Velocity X               KPS
      orbital_state_velocity.Vec[0] flt64 64          "appid18,pktid1"
      320

                        5           ORBSTATEVELY      Point to
propagate from - Velocity Y               KPS
      orbital_state_velocity.Vec[1] flt64 64          "appid18,pktid1"
      384

                        6           ORBSTATEVELZ      Point to
```

```
propagate from - Velocity Z              KPS
      orbital_state_velocity.Vec[2] flt64 64          "appid18,pktid1"
      448

                              7          ORBSTATETIME      Time to
propagate from                SEC
      orbital_state_time      flt64 64          "appid18,pktid1"  1728
```

Code Listing 3-2: Example Commands in Plain Text Format

```
mission_id  mission     object_id   object_path tlm_id      old_name
      new_name    desc  data_type   bit_length  EU    range conv_to_EU
      flight_DN_EU_conv monitor_limits
"DATABASE MISSION ID (if available)"      "MISSION NAME"    "DATABASE
OBJECT ID (if available)"     "COMMA SEPARATED LIST OF PATH TO OBJECT"
      "DATABASE TELEMETRY ID (if available)"    "OLD TELEMETRY NAME
(required if updating existing data)"     "NEW TELEMTRY NAME (required
for new telemetry)"     "TELEMETRY ITEM DESCRIPTION"  "TELEMETRY ITEM
DATA TYPE"  "NUMBER OF BITS IN TELEMETRY ITEM"  "ENINEERING UNIT (if
applicable)"      "RANGE OF VALUES" "LIST OF PARAMETERIZED COEFFICEINTS
USED TO CONVERT DN TO EU"     "FALSE IF DN TO EU CONVERSION IS DONE BY
GROUND SYSTEM"    "MONITOR LIMITS (if applicable)"
```

Code Listing 3-3: Telemetry Template in Plain Text Format

```
mission_id  mission     object_id   object_path tlm_id      old_name
      new_name    desc  data_type   bit_length  EU    range conv_to_EU
      flight_DN_EU_conv monitor_limits    pkt_id      pkt_path
      start_bit_in_packet

      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[0]
"                 ADFORBSTATEPOSX   Progagated Point 1 position X flt64
      64    KM          "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.0,"   FALSE             "appid18,pktid2"  112
```

Code Listing 3-4: Example Telemetry in Plain Text Format

If the data comes in the form of an XML flat file, it must be placed in the following format for commands (see Code Listing 3-5):

```
<database>DATABASE NAME
      <mission>MISSION NAME
            <mission_id>DATABASE MISSION ID (if available)</mission>
            <cmd>
                  <object_id>DATABASE OBJECT ID (if available)
                        </object_id>
                  <object_path>COMMA SEPARATED LIST OF PATH TO OBJECT
                        </object_path>
                  <cmd_id>DATABASE CMD ID (if available)</cmd_id>
                  <old_cmd_name>OLD CMD NAME (required if updating
                        existing data)</old_cmd_name>
                  <new_cmd_name>NEW CMD NAME (required for new
                        commands)</new_cmd_name>
                  <cmd_desc>CMD DESCRIPTION</cmd_desc>
                  <cmd_type>COMMAND TYPE</cmd_type>
                  <cmd_bit_length>NUMBER OF BITS IN CMD
                        </cmd_bit_length>
                  <delay>REQUIRED DELAY (if required)</delay>
                  <predecessor>NOTES ABOUT ANY CMDS THAT MUST BE RUN
                        BEFORE THIS CMD</predecessor>
                  <is_critical>TRUE OR FALSE</is_critical>
                  <cmd_constraints>CMD CONSTRAINTS INFO
                        </cmd_constraints>
                  <resultant_state>RESULTING STATE INFO
                        </resultant_state>
                  <tlm_response>ADD ANY NUMBER OF TLM RESPONSES
                    <tlm_id>DATABASE TLM ID (if available)</tlm_id>
                    <tlm_name>TLM NAME</tlm_name>
                  </tlm_response>
                  <cmd_param>ADD ANY NUMBER OF CMD PARAMETERS
                        <cmd_param_id>DATABASE CMD PARAM ID (if
                              available)</cmd_param_id>
                        <param_number>ORDER OF PARAMETER WITHIN THE CMD
                              </cmd_number>
                        <old_param_name>OLD PARAMETER NAME (required if
                              updating existing data</old_param_name>
                        <new_param_name>NEW PARAMETER NAME (required
                              for new parameters)</new_param_name>
                        <param_desc>PARAMETER DESCRIPTION</param_desc>
                        <default_value>DEFAULT VALUE (if available)
                              </default_value>
                        <value>SPECIFIC VALUE (if available)</value>
                        <EU>ENGINEERING UNITS</EU>
                        <input_type></input_type>
                        <conv_to_DN>CONVERSION TO DATA NUMBER (if
                              needed)</conv_to_DN>
                        <range>ALLOWABLE RANGE FOR PARAMETER VALUE
                              </range>
                        <data_type>PARAMETER DATA TYPE</data_type>
                        <param_bit_length>NUMBER OF BITS IN PARAMETER
                              </param_bit_length>
                        <ANY CUSTOM PARAMETER ATTRIBUTES></ANY CUSTOM
                              PARAMETER ATTRIBUTES>
                        ...
                  </cmd_param>
                  <ANY CUSTOM COMMAND ATTRIBUTES></ANY CUSTOM COMMAND
                        ATTRIBUTES>
                        ...
            </cmd>
      </mission>
</database>
```

Code Listing 3-5: Command Template in XML Format

```
<database>CTDB
      <mission>DEEP IMPACT
            <cmd>
                  <object_id></object_id>
                  <object_path>DI,Flyby,ADF,APPID18,PKTID1
                        </object_path>
                  <old_cmd_name></old_cmd_name>
                  <new_cmd_name>ADFOP_INPUTS</new_cmd_name>
                  <cmd_desc>Set orbit propagation inputs</cmd_desc>
                  <cmd_type>SWC</cmd_type>
                  <cmd_bit_length>2024</cmd_bit_length>
                  <delay></delay>
                  <predecessor></predecessor>
                  <is_critical>FALSE</is_critical>
                  <cmd_constraints></cmd_constraints>
                  <tlm_response>
                    <tlm_id></tlm_id>
                    <tlm_name></tlm_name>
                  </tlm_response>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>1</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEPOSX</new_param_name>
                        <param_desc>Point to propagate from - Position
                              X</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KM</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>
                        <fsw_var_name>orbital_state_position.Vec[0]
                              </fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>0</pkt_start_bit>
                  </cmd_param>
            </cmd>
      </mission>
</database>
```

Code Listing 3-6: Example Commands in XML Format

And be placed in the following XML format for telemetry (see Code Listing 3-7):

```
<database>DATABASE NAME
      <mission>MISSION NAME
            <mission_id>DATABASE MISSION ID (if available)</mission_id>
            <tlm>
                  <object_id>DATABASE OBJECT ID (if available)
                        </object_id>
                  <object_path>COMMA SEPARATED LIST OF PATH TO
                         OBJECT</object_path>
                  <tlm_id>DATABASE TELEMETRY ID (if available)</tlm_id>
                  <old_name>OLD TELEMETRY NAME (required if updating
                        existing data)</old_name>
                  <new_name>NEW TELEMTRY NAME</new_name>
                  <desc>TELEMETRY ITEM DESCRIPTION</desc>
                  <data_type>TELEMETRY ITEM DATA TYPE</data_type>
                  <bit_length>NUMBER OF BITS IN TELEMETRY ITEM
                        </bit_length>
                  <EU>ENINEERING UNIT (if applicable)</EU>
                  <range>RANGE OF VALUES</range>
                  <conv_to_EU>LIST OF PARAMETERIZED COEFFICEINTS USED
                         TO CONVERT DN TO EU</conv_to_EU>
                  <flight_DN_EU_conv>FALSE IF DN TO EU CONVERSION IS
                         DONE BY GROUND SYSTEM</flight_DN_EU_conv>
                  <monitor_limits>MONITOR LIMITS (if applicable)
                        </monitor_limits>
                  <ANY CUSTOM TELEMETRY ATTRIBUTES>
                        </ANY CUSTOM TELEMETRY ATTRIBUTES>
                  ...
            </tlm>
      </mission>
</database>
```

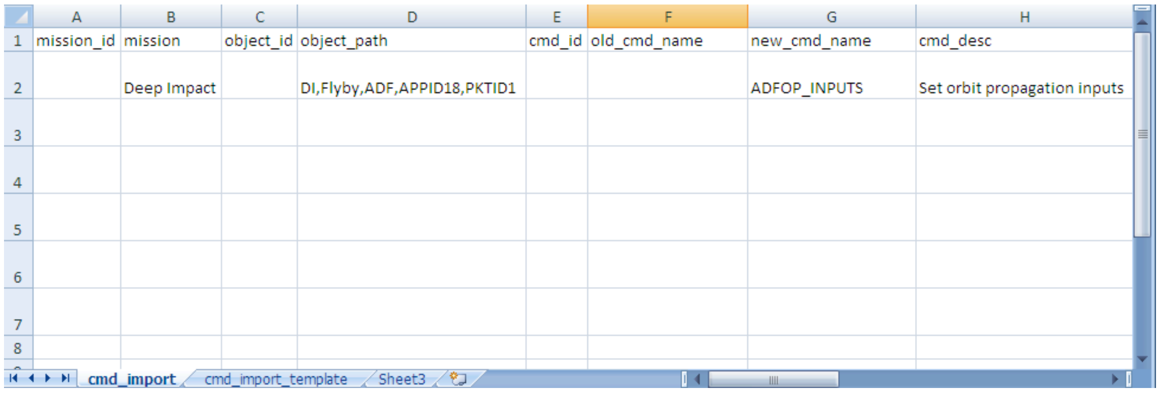
Code Listing 3-7: Telemetry Template in XML Format

```
<database>CTDB
      <mission>Deep Impact
            <mission_id>1</mission_id>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_position, Vec[0]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEPOSX</new_name>
                  <desc>Progagated Point 1 position X</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KM</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
```

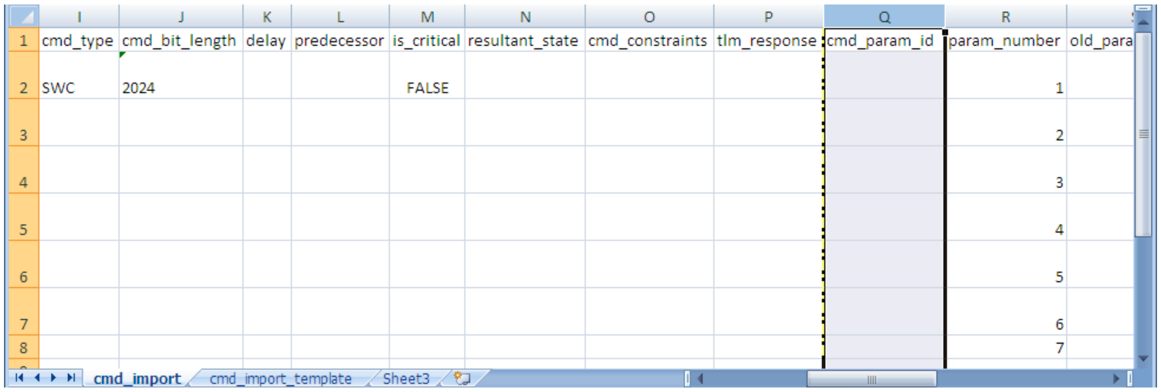
```
                <pkt_start_bit>112</pkt_start_bit>
            </tlm>
            <tlm>
                <object_id></object_id>
                <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_position, Vec[1]</object_path>
                <tlm_id></tlm_id>
                <old_name></old_name>
                <new_name>ADFORBSTATEPOSY</new_name>
                <desc>Progagated Point 1 position Y</desc>
                <data_type>flt64</data_type>
                <bit_length>64</bit_length>
                <EU>KM</EU>
                <range></range>
                <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                <flight_DN_EU_conv>false</flight_DN_EU_conv>
                <monitor_limits></monitor_limits>
                <pkt_id></pkt_id>
                <pkt_path>appid18,pktid2</pkt_path>
                <pkt_start_bit>176</pkt_start_bit>
            </tlm>
        </mission>
</database>
```

Code Listing 3-8: Example Telemetry in XML Format

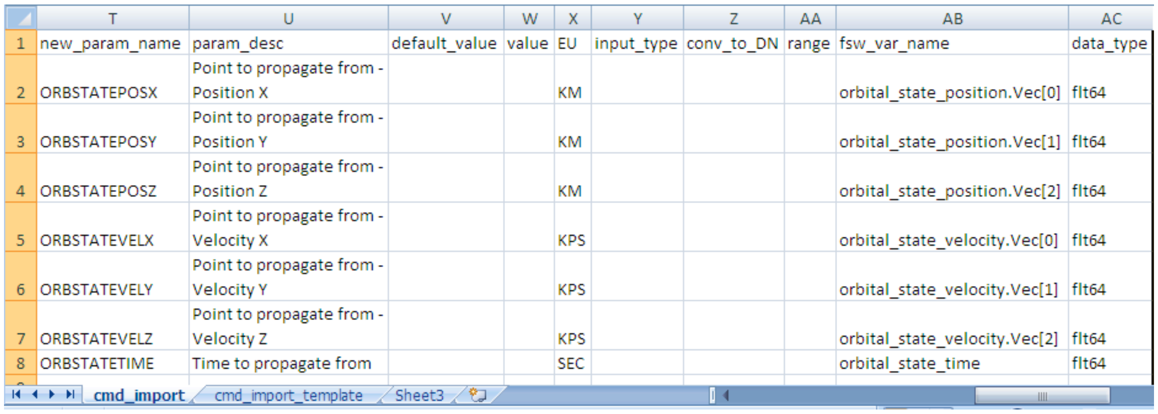
The final option allowed is to import data through a Microsoft Excel spreadsheet file (2003 version). The contents of each spreadsheet show below have been broken into a series of snapshots due to the width of the spreadsheet.



	A	B	C	D	E	F	G	H
1	mission_id	mission	object_id	object_path	cmd_id	old_cmd_name	new_cmd_name	cmd_desc
2		Deep Impact		DI,Flyby,ADF,APPID18,PKTID1			ADFOP_INPUTS	Set orbit propagation inputs
3								
4								
5								
6								
7								
8								

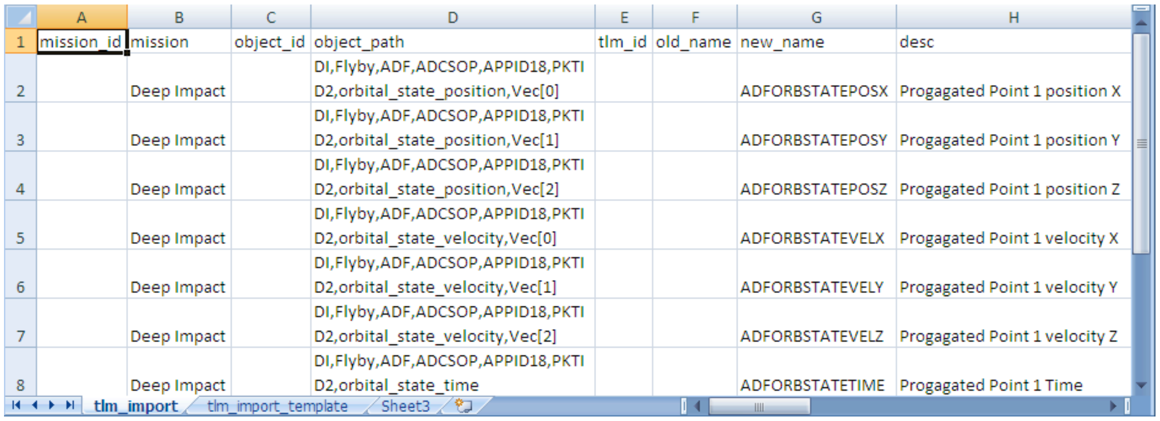


	I	J	K	L	M	N	O	P	Q	R	S
1	cmd_type	cmd_bit_length	delay	predecessor	is_critical	resultant_state	cmd_constraints	tlm_response	cmd_param_id	param_number	old_para
2	SWC	2024			FALSE					1	
3										2	
4										3	
5										4	
6										5	
7										6	
8										7	

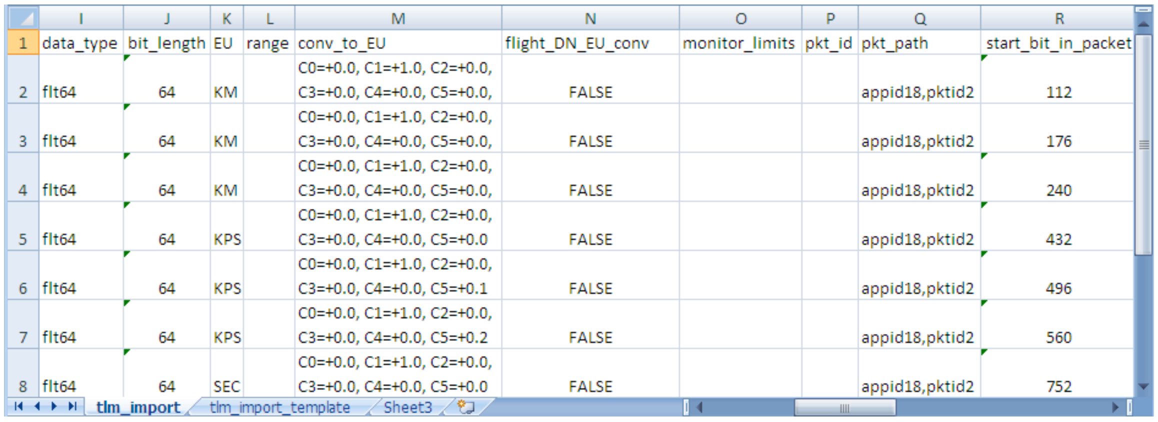


	T	U	V	W	X	Y	Z	AA	AB	AC
1	new_param_name	param_desc	default_value	value	EU	input_type	conv_to_DN	range	fsw_var_name	data_type
2	ORBSTATEPOSX	Point to propagate from - Position X			KM				orbital_state_position.Vec[0]	flt64
3	ORBSTATEPOSY	Point to propagate from - Position Y			KM				orbital_state_position.Vec[1]	flt64
4	ORBSTATEPOSZ	Point to propagate from - Position Z			KM				orbital_state_position.Vec[2]	flt64
5	ORBSTATEVELX	Point to propagate from - Velocity X			KPS				orbital_state_velocity.Vec[0]	flt64
6	ORBSTATEVELY	Point to propagate from - Velocity Y			KPS				orbital_state_velocity.Vec[1]	flt64
7	ORBSTATEVELZ	Point to propagate from - Velocity Z			KPS				orbital_state_velocity.Vec[2]	flt64
8	ORBSTATETIME	Time to propagate from			SEC				orbital_state_time	flt64

Figure 3-1: Example Command Data in Excel Spreadsheet Format



	A	B	C	D	E	F	G	H
1	mission_id	mission	object_id	object_path	tlm_id	old_name	new_name	desc
2		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[0]			ADFORBSTATEPOSX	Progagated Point 1 position X
3		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[1]			ADFORBSTATEPOSY	Progagated Point 1 position Y
4		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[2]			ADFORBSTATEPOSZ	Progagated Point 1 position Z
5		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[0]			ADFORBSTATEVELX	Progagated Point 1 velocity X
6		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[1]			ADFORBSTATEVELY	Progagated Point 1 velocity Y
7		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[2]			ADFORBSTATEVELZ	Progagated Point 1 velocity Z
8		Deep Impact		DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_time			ADFORBSTATETIME	Progagated Point 1 Time



	I	J	K	L	M	N	O	P	Q	R
1	data_type	bit_length	EU	range	conv_to_EU	flight_DN_EU_conv	monitor_limits	pkt_id	pkt_path	start_bit_in_packet
2	flt64	64	KM		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0,	FALSE			appid18,pktid2	112
3	flt64	64	KM		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0,	FALSE			appid18,pktid2	176
4	flt64	64	KM		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0,	FALSE			appid18,pktid2	240
5	flt64	64	KPS		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0	FALSE			appid18,pktid2	432
6	flt64	64	KPS		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.1	FALSE			appid18,pktid2	496
7	flt64	64	KPS		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.2	FALSE			appid18,pktid2	560
8	flt64	64	SEC		C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0	FALSE			appid18,pktid2	752

Figure 3-2: Example Telemetry Data in Excel Spreadsheet Format

3.3 Parsing the Data

Once the import files conform to the minimum necessary standards established above, the data is parsed into a database "staging area," a series of temporary data tables in which the data can reside while the storage framework performs additional import process steps such cleaning and verifying the quality of the data. In order to transfer data into this staging area, the import data must pass through a parsing code designed to locate the applicable file or files, open and read it from any of the available file formats, and subsequently enter the data values into the appropriate locations within the staging tables. Upon completion, this parsing results in a fully populated set of staging tables data. These temporary tables are identical in structure, though perhaps not in content, for both full and partial imports. It should be noted that multiple sets of staging data can be present in the staging tables, where each set refers to all of the data pertinent to a specific import. A unique import execution id is assigned to each set, allowing the framework software to differentiate between data sets. This is import for the case of partial or incremental bulk imports as more than one can be concurrently executing by different users. The details of this are discussed in Section 3.5.

These staging tables provide the first user feedback, presenting the data in a format that allows the user to quickly identify gross errors, such a wrong file format, unreadable data, or required fields left incomplete. Thus, if gross errors are present, the user can choose to fix the error and restart the importation before progressing any further, potentially saving a great deal of time and frustration. This represents the first, most basic level of error checking for this database system.

3.4 Data Quality Control

Having placed the incoming data into these staging tables, we are now in a position to comprehensively validate the correctness of the data, required primarily due to the prevalence of human error in defining the contents of the bulk import files. This step allows a group to be highly confident that the data entered into the storage framework is indeed what was intended. The validation process will allow us to determine that the data meets internal data integrity constraints, as well as any imposed business constraints. At this point in the importation procedure, the system has no real assurance that the incoming data constitutes proper commands and telemetry; the validation process, as it is carried out against the data in the staging tables, ensures that this data is meaningful and that it can be successfully inserted into the rest of the database, while simultaneously enforcing that any such changes will not corrupt the data that is already present. The validation process accomplishes this by running a series of checks that first verify that the incoming data meets the standards established for commands and telemetry in such terms as including all required fields, secondly, that the data conforms to any other internal database constraints that have been implemented, thirdly, meets any requirements of established and user-defined business rules, and finally, that the data does not violate the "skeleton" object hierarchy. All of these checks are attempts to avoid or reduce the infamous "garbage in, garbage out" problem of database usage. This step also provides a means of tracking progress towards completing of all of the checks, report problems to the user as they arise, and can subsequently allow the process to be restarted after corrective action, potentially saving a great deal of time and frustration on the part of the data input users.

The implementation of these checks can typically be done either as part of the parsing code as it is reading the data or, as I have implemented it here, as a separate step once the data has been entered into the database and can be accessed through SQL functionality. Implementing data quality control checks in the parsing code can have the benefits of increased flexibility and ease of future alteration, and it does not require that the entire pertinent data set be re-imported whenever any significant errors occur. However, it is much more difficult to have access to enough data context to perform more comprehensive tests on the new data set when these checks are implemented only in the parsing code. Additionally, the data validation process usually cannot be restarted easily or from multiple points, instead requiring that the process be rerun from the beginning. Having the data already in a staging table allows us to take advantage of fast SQL queries against the entire imported data set in conjunction with access to already existing data, and also provides a convenient storage mechanism for instances when errors are detected. The data does not have to be reread; instead, the user can respond and make changes in the staging tables, sometimes just by re-importing the affected data, in which case the process can be restarted from the last failed check. But the primary benefits to running these data validation checks against the staging tables is the access to context (both within the new dataset and the existing data) and the persistence of the data, i.e. the user's ability to return to the data at any given time; many processes can run for a period of hours, or even days, and by placing the data in staging tables, the user need not monitor the progress constantly, but can return to correct errors the next working day.

If both commands and telemetry are included in a given import, data validation first takes place against the data in the telemetry staging tables. The data validation process for a telemetry item is fairly brief. The framework first logs the beginning of the telemetry validation process and reads the first record in the telemetry staging table, verifying that all required data is present and that a valid hierarchy item exists with which to associate the telemetry point. The framework then marks the individual telemetry record as valid. Should the record contain errors, a message detailing the errors will be appended to the record. This same process is repeated with all top-level telemetry items, and at the conclusion of processing each staging table, the framework logs the concluding time for the process. At the conclusion, all records containing errors are reported to the user. If the user specified a table-by-table report, the error notes will be collected and presented before the program moves from one table to the next; otherwise, the error notes will be presented at the conclusion of the entire data validation process. The user can then correct the errors either directly in the table or, should s/he choose, in the original file and re-import the data segment in question. This process is repeated for each ancillary telemetry staging table, with the additional constraint that it must be able to be associated with a validated telemetry point in the primary telemetry staging table.

The validation process for commands is relatively similar to that of telemetry: it logs the start of the command validation process, reads through each record of the command staging table, checks that each record meets all constraints and that it is associated with a hierarchy object, marks each record as valid or as containing errors, and generates a report back to the user at a time specified by the user. Unlike the telemetry process, however, command changes require the modification of two primary staging tables: the command table (CT) and the command-parameter table (CPT). The CPT is validated after the CT, and includes the additional check that each point must be associated with a previously validated command. After the CT and CPT have been validated, the framework moves on to the ancillary command tables and finally to the command-telemetry response table (CTRT), any of which may or may not contain data. Any ancillary tables are validated in a similar manner to the CPT; the CTRT has the additional requirement that each point must be associated both with a validated command and with a telemetry record. Due to this constraint, the CTRT staging table will always be processed last.

Once all of the imported data has been validated to user's and the system's satisfaction, the user can then finalize the importation, overwriting any preexisting data and making the new data available to all users.

3.5 Full and Incremental Updates

The importation of valid data can occur in two forms: a full and complete overwrite of an entire set of commands and telemetry, or a partial overwrite of a specified section of the mission object hierarchy. Any spacecraft command and telemetry storage framework built with the idea of practical usability must include both of these options. In general, a bulk import of any size allows a user to make a large number of changes at one time, but in some cases in which only individual sections need revision, an overwrite of the entire database would waste valuable time, and would introduce possibilities for inadvertent changes in data that lies outside the intended alterations. Allowing for partial, or incremental, updates obviates these problems and also allows for the division of responsibility among team members, for example allowing users to deal with instruments and payloads separately from spacecraft while still ensuring mutual compatibility. Having the option of incremental updates also allows multiple users to alter the database concurrently. (Multiple user interfaces will be discussed further in Chapter 4).

In general, for a full import, the entire existing data set has been exported, modified, and then re-imported, replacing all command and telemetry information. The entire database is first locked down for the duration of the import, meaning no new changes can be made by any user. A backup is made of the original database after which all command and telemetry fields are deleted and then reinstated in a predetermined order. If the user is notified about an error in the newly reconstituted database, the user then has the option of correcting the error and restarting the import process or else the user can restore the original dataset from the backup.

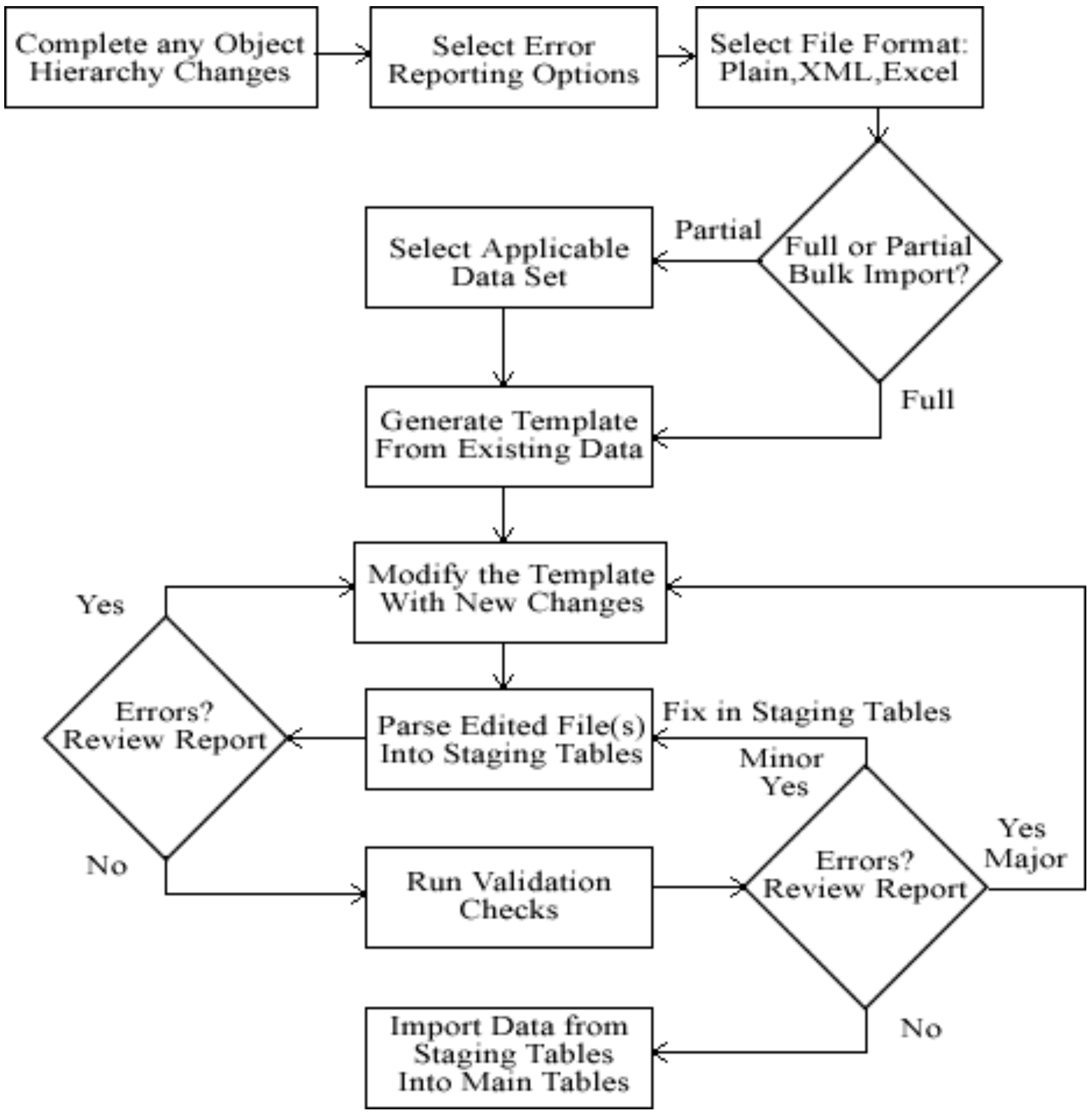

Complete any Object Hierarchy Changes
Select Error Reporting Options
Select File Format: Plain,XML,Excel
Full or Partial Bulk Import?
Partial
Full
Select Applicable Data Set
Generate Template From Existing Data
Modify the Template With New Changes
Yes
Errors? Review Report
No
Parse Edited File(s) Into Staging Tables
Fix in Staging Tables
Minor Yes
Yes Major
Run Validation Checks
Errors? Review Report
No
Import Data from Staging Tables Into Main Tables

Figure 3-3: Command and Telemetry Import Decision Flow Chart

More complicated, however, is the process of an incremental update. In this case, the user must first specify which portion of the object hierarchy (see Chapter 2) is to be targeted for modification of its associated command and/or telemetry data. Any pre-existing command and telemetry data for that portion is included in the generated template file in order to help provide a basis for user revision. The helps to avoid requiring users to recreate the data from scratch each time if their goal was simply to revise or expand the pre-existing data rather than to define completely new definitions. In contrast to a full import, only command and telemetry records associated with the targeted hierarchical section are locked from editing throughout the duration of the import. The section to be updated is, as in a full update, backed up. During the validation step before the section is rewritten, however, the system runs an additional series of checks in order to identify other elements of the database that are affected by, though not directly encompassed within, the new upload; the system then warns the user about these additional affected sections. Should the system be able to resolve the conflicts between associated sections it will do so, but in most cases the associated data will be deleted in the process of the update. This warning serves, therefore, merely as a checkpoint, alerting the user to the potential errors resulting from the update, and forcing the user to confirm his/her desire to proceed with the process. For example, if a user is updating or removing a telemetry item that has been associated with a command-telemetry response record (which is normally handled during command imports); if it is incapable of resolving any conflicts (such as be re-associating the command with a new or updated telemetry record), the system will alert the user that the applicable command-telemetry record will be deleted.

Incremental updates pose a particular problem in the area of restoration. Should a user, having perceived a problem in the newly uploaded data, choose to revert to the original dataset, s/he cannot restore directly from a typical full backup of the database, due to the fact that other changes may have occurred outside areas that have been locked down. Therefore, the incremental backup step has two components, the first being the creation of a typical full database backup to provide a robust safety net in case of full-scale disaster, and secondly, the system internally stores an additional copy of the same exported dataset file that was given to the user. This duplicate, which preserves the originally selected dataset (and is never modified), can be used to restore the affected portion of the database. The system does this by automatically re-running the incremental import process on that duplicate.

Chapter 4

Multi-User Environment

4.1 Overview

In reality, any mission involving spacecraft requires a team of people working in parallel, not in series, which raises the issue of concurrent user access to the data. In some cases, teams choose to address these problems through the assignation of the data-change administrator role to a single team member. Such a person is responsible for collecting all requests for data changes and implementing them in such a way as to avoid conflict and the introduction of data corruption. This "single user as checkpoint" is certainly implementable, but it is neither time nor cost effective. Multiple users often must wait for long periods of time, so that previously requested changes can be implemented, before continuing on with new work. To avoid these issues, my framework allows for multi-user access to the database through a system of access controls, allowing different team members to work concomitantly and yet preserving data integrity through the assignation of roles.

4.2 Implementation

In general, the database is designed to be operated with open access in the case of real-time updates through web interfaces (i.e. "last change in wins"); such real-time alterations are limited to individual data records and thus do not usually interfere excessively with other user access. If a particular group feels that this open access poses issues to data integrity, the group can self-enforce regulating procedures. The system also tags each record with the identity of the user making changes and the time at which those changes occurred, making tracking of alterations feasible. In the case of file-based imports, however, the database is a closed system. A full import locks down the entire database from changes, while tailored subsets of the hierarchy lock down during partial imports (see section 4). Beyond these locking mechanisms, however, the database provides a second level of access control. This system requires an assignation of role to each user, which limits a given user's ability to view or update records and restricts the types of updates allowed. Individual users can be assigned any of the following roles:

- View only (guest access)
- and edit through real-time web-based interface (single record access)
- and edit via partial imports (single hierarchical subset access)
- and edit via full import (full hierarchical access)
- and customize system through accessing tables, setting up users, and assigning role (administrative)

role_id	role_name	role_desc
1	view	User can view all data and generate reports but cannot edit
2	edit	View role plus editing through web interface
3	partial import	Edit role plus ability to perform partial bulk imports
4	full import	Partial import role plus ability to perform full bulk imports
5	user admin	Full import role plus ability to add/modify other users
6	system admin	Unrestricted privileges; can run maintenance functionality

Table 4-1: t_role Table With Data

user_id	first_name	last_name	email	username	password
1	Trusty	Dogg		tbone	asdfJ33a_Ad
2	Comm	Pewter		ithink	7asdf—aJnd
3	Mya	Supervisor		mya	Vdavvd8dh=

Table 4-2: t_user Table with Example Data

user_id	mission_id	role_id
1	1	1
1	2	4
2	1	2
3	1	3

Table 4-3: t_user_mission_role_lookup Table with Example Data

In the example data listed in Tables 4-1 through 4-3, we can see that the user Trusty Dogg has privileges to the data for two missions. For mission_id 1, Trusty has the view only role, restricting him from modifying any data. However, for mission_id 2, Trusty has full import privileges. Two other users, Comm and Mya, have also been assigned roles for mission_id 1 data. Therefore we can see that whenever multiple missions are simultaneously stored within a given instance of this storage framework, the lookup table, hence "t_user_mission_role_lookup," becomes even more significant. In addition to identifying which role is associated with a given user, this table tracks which mission data that user has access to. A user can potentially have different roles for different missions. All framework functionality being run for a user is verified in software and any SQL queries or stored procedures against what is allowed by their particular role and mission privileges.

Aside from preserving data integrity during the process of database construction, this assignation of roles also allows administrators to redefine users' roles as the status of the system changes. Should the database ever be "complete," either as a whole or in part, for example, the administrator can redefine user access to “View Only” to prevent further changes.

Chapter 5

Conclusion

Having introduced the topic of spacecraft command and telemetry storage frameworks, presented examples of current custom, in-house, and COTS solutions, and presented the various issues involved with these systems, I have identified a few key areas that can consistently result in a measurable change to a mission's schedule, cost, and usability. They are:

- Reuse: The ability to adapt the same storage framework for use across multiple missions and spacecraft. Typically these command and telemetry storage frameworks are custom built for each mission or spacecraft, and sometimes even by each party involved for their own specific portion of the overall mission. Various parties may have made inroads on an in-house storage system, but these reusable savings are generally lost at higher levels of integration.
- Flexibility: The ability of the storage framework to model the real needs of the mission, both as a means to manage the evolution of the data over the life-cycle of the mission, and as a means to integrate data from different groups or users into an authoritative repository in an ongoing basis. Without an adequately flexible storage system, designers cannot accommodate any unforeseen changes in the late mission stages.
- Usability: How the data moves from varied sources into the storage framework and back out in a robust, traceable, and repeatable manner that does not place an

unreasonable burden on the users. The lack of adequate usability can be detrimental to the framework's practical use and to the mission schedule and cost.

5.1 Work Completed

This paper has presented a high-level view of implementation solutions that address the above areas in order to create the foundation of a practical, reusable command and telemetry storage framework. I have specifically researched, addressed, and implemented three framework components: first, a flexible 'skeleton' to organize the data; secondly, a robust file-based bulk import process; and finally, an overview of managing multi-user concurrent data access within the framework in terms of roles.

5.2 Future Work

In order to create a complete framework, much further research and design is required, specifically, the implementation of a secure online distributed web services interface and a programmatic data access and report generation API. The web interface should function as the primary point of entry into the storage framework and should provide a number of controls, including mission and user setup and management; minor editing through online forms (i.e. anything for which bulk imports are excessive); data browsing and report generation; and online help manuals to the average user. The API should provide a flexible means to place generated artifacts within the context of various software tool chains employed by the spacecraft mission design and operation. The API should be extended to support data translation to and from the OMG XTCE format to allow for industry-wide and standardized data transfer and usage to other third party framework components.

References

[1] SHIM, J-M (2006), “Korean geostationary satellite programme: Communication, Ocean and meteorological Satellite (COMS),” in The 2006 EUMETSAT Meteorological Satellite Conference, EUMETSAT, p. 48.

[2] McQUEARY, L. (1998), “Middleware in COTS Command and Control Architectures,” in Ground System Architecture Workshop 1998, The Aerospace Corporation, available at http://sunset.usc.edu/GSAW/gsaw98/agenda98.html/.

[3] DUBON, L. P. (2006), “From zero to integration in eight months, the Dawn Ground Data System engineering challenge,” AIAA 9th International Conference on Space Operations (SpaceOps), JPL and NASA, available at http://hdl.handle.net/2014/39806.

[4] RAYMAN, M.D., T.C. FRASCHETTI, C.A. RAYMOND, and C.T. RUSSELL (2006), “Dawn: a mission in development for exploration of mainbelt asteroids Vesta and Ceres,” Acta Astronautica 58, pp. 605-616.

[5] HAWES, D. (2007), “Mission Operations Status,” for IBEX from the Jet Propulsion Laboratory, available at http://ibex.swri.org/multimedia/discuss/c_ops.pdf/ last visited April, 2009.

[6] KINLI, C. A., M. GAMACHE, M. ROSE, A. ROST, J. SALES, and J. TANG (2004), “Telemetry, Tracking, Communications, Command and Data Handling,” available online at http://www.aoe.vt.edu/~cdhall/courses/aoe4065/FDReports/CDH04.pdf/ last visited April, 2009.

[7] Online article “XML Telemetric & Command Exchange (XTCE)” from Object Management Group available at http://www.omg.org/space/xtce/ last visited April, 2009.

[8] LEE, S, J. WON CHAN and J-H KIM (2008), “Task Scheduling Algorithm for the Communication, Ocean, and Meteorological Satellite,” ETRI 30 (1), pp. 1-12.

[9] Online proprietary information for OS/COMET from Harris Corporation available at http://www.govcomm.harris.com/oscomet/products/main_product_suite.asp/, last visited April, 2009.

[10] Online proprietary information for InControl-NG from L-3 Communications Telemetry-West available at http://www.l-3com.com/tw/, last visited April, 2009.

[11] Online press release “L-3 Storm Control Systems to Provide Satellite Test Software Solution for Advanced Extremely High Frequency (AEHF) Program”

(2006) from L3 Communications available at http://www.l-3com.com/TW/telemetry_west/news/scouts.htm/ last visited April, 2009.

[12] Online article “Missions Data System” (2007) from Jet Propulsions Laboratory available at http://mds.jpl.nasa.gov/ last visited January, 2009.

[13] Online article “Deep Impact: mission to a Comet” (2008) from NASA available at http://www.nasa.gov/mission_pages/deepimpact/main/index.html/ last visited January, 2009.

[14] RASMUSSEN, R. D., D. DVORAK, K. GOSTELOW, T. STARBIRD, E. GAT, S. CHIEN et al (2004), “U.S. Patent 6745089,” available at http://www.freepatentsonline.com/6745089.html/ last visited January, 2009.

[15] STEPHENS, R. K. and R. R. PLEW (2000), *Database Design*, Sams Publishing, Indianapolis.

[16] Online database JDBC Basics: The Java Tutorials from Sun Microsystems, Inc., available at http://java.sun.com/docs/books/tutorial/jdbc/basics/ last visited January, 2009.

[17] Online article “Managing Hierarchical Data in MySQL” from Sun Microsystems, Inc., available at http://dev.mysql.com/tech-resources/articles/hierarchical-data.html/ last visited January, 2009.

[18] COOPER, J. W. (2000) Java Design Patterns: A Tutorial, Addison-Wesley, Reading, Mass.

[19] BENNET, M., R. RASMUSSEN, D. DVORAK, J. MORRIS, M. INGHAM and D. WAGNER (2007), State Analysis for Software Engineers available at https://pub-lib.jpl.nasa.gov/docushare/dsweb/View/Collection-68/ last visited January, 2009.

[20] EDWARDS, C. D., B. ARNOLD, R. DePAULA, G. KAZZ, C. LEE and G. NOREEN (2006) “Relay communications strategies for Mars exploration through 2020,” Acta Astronautics, 59, pp. 310-318.

[21] Online article “Technical Support Package for Extraction and Analysis of Display Data,” NASA Tech Briefs, MSC-23630-1, available at http://www.techbriefs.com/component/content/article/3352/ last visited January, 2009.

[22] MINGER, S. and T. GENRICH (2008), “Programmable Satellite Transceiver for Responsive Space,” Aerospace Conference, 2008 IEEE, pp. 1-12.

[23] BARKLEY, E. and P. PECHKAM (2007), “CCSDS Space Link Extension Service Management: Real-World Use Cases,” AIAA 220, pp. 101-117.

[24] BAGRI, D. S., J. I. STATMAN and M. S. GATTI (2005), “Operation’s Concept for Array-based Deep Space Network,” Aerospace Conference, 2005 IEEE, pp. 1532-1540.

[25] Online “D-Command and Telemetry Software-Support for Launch Control System” available at https://www.fbo.gov/ last visited January, 2009.

[26] VINTERHAV, E. and T. KARLSSON (2007) “Script based software for ground station and mission support operations for the Swedish small satellite Odin,” Acta Astronautica, 61, pp. 912-922.

[27] PFARR, B., J. DONOHUE, B. LUI, G. GREER and T. GREEN (2008), “Proven and Robust Ground Support Systems - GSFC Success and Lessons Learned,” Aerospace Conference, 2008 IEEE, pp. 1-7.

[28] GAL-EDD, J. and C. C. FATIG (2006), “James Webb Space Telescope XML database: from the beginning to today,” Aerospace Conference, 2006 IEEE, pp. 1-7.

[29] CHEVERS, D. D., D. DURHAM, and T. ITCHKAWICH (2006), “MAESTRO: the versatile command and control system software for mission operations and testing,” Aerospace Conference, 2006 IEEE, pp. 1-8.

[30] SNIFFIN, R. W. (2000), *DSN Telecommunications Link Design Handbook*, Jet Propulsion Laboratory, Pasadena.

[31] Online interview with Dan Smith (May 2007) available at http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070036796_2007034178.pdf last visited January, 2009.

[32] Online press release, “NASA Taps Harris’ Software for Constellation Launch Control Program” (2008), COTS Journal, available at http://www.cotsjournalonline.com/home/article.php?id=100878/ last visited April, 2009.

[33] Online press release, “NASA's Launch of Carbon-Seeking Satellite is Unsuccessful” (2009), Orbiting Carbon Observatory site from the Jet Propulsion Laboratory, available at http://oco.jpl.nasa.gov/ last visited March, 2009.

[34] DeWITT, S. and VOILAND, A., online press release, “NASA's Next Climate-Research Satellite One Step Closer to Orbit” (2009), available at http://www.nasa.gov/centers/goddard/news/topstory/2009/aps_test.html last visited March, 2009.

[35] *Managing the Use of Commercial Off the Shelf (COTS) Software Components for Mission-Critical Systems* (2006), American Institute of Aeronautics and Astronautics, Reston, Virginia.

Appendix A

Sample Hierarchy Configuration File: HRI_hierarchy_config.xml

```
<database>CTDB
      <mission>Deep Impact
            <description>NASA's Deep Impact comet cratering mission -
exploring the origins of our solar system.</description>
            <object>Flyby Spacecraft
                  <description>The main transport
spacecraft</description>
                  <object>HRI
                        <description>High Resolution Imager
Instrument</description>
                        <object>filter wheel 1
                              <description>color filters</description>
                        </object>
                        <object>filter wheel 2
                              <description>saturation
filters</description>
                        </object>
                  </object>
            </object>
      </mission>
</database>
```

Sample Command File – Text Format: cmd_import.txt

```
mission_id  mission     object_id   object_path cmd_id
      old_cmd_name      new_cmd_name      cmd_desc    cmd_type
      cmd_bit_length    delay predecessor is_critical resultant_state
      cmd_constraints   tlm_response      cmd_param_id
      param_number      old_param_name    new_param_name    param_desc
      default_value     value EU    input_type  conv_to_DN  range
      fsw_var_name      data_type   param_bit_length  pkt_id
      pkt_path    pkt_start_bit
      Deep Impact       "DI,Flyby,ADF,APPID18,PKTID1"
      ADFOP_INPUTS      Set orbit propagation inputs  SWC   2024
      FALSE                         1           ORBSTATEPOSX      Point
to propagate from - Position X                KM
      orbital_state_position.Vec[0] flt64 64          "appid18,pktid1"
      0

                              2           ORBSTATEPOSY      Point to
propagate from - Position Y               KM
      orbital_state_position.Vec[1] flt64 64          "appid18,pktid1"
      64

                              3           ORBSTATEPOSZ      Point to
propagate from - Position Z               KM
```

```
      orbital_state_position.Vec[2] flt64 64          "appid18,pktid1"
      128

                              4           ORBSTATEVELX      Point to
propagate from - Velocity X               KPS
      orbital_state_velocity.Vec[0] flt64 64          "appid18,pktid1"
      320

                              5           ORBSTATEVELY      Point to
propagate from - Velocity Y               KPS
      orbital_state_velocity.Vec[1] flt64 64          "appid18,pktid1"
      384

                              6           ORBSTATEVELZ      Point to
propagate from - Velocity Z               KPS
      orbital_state_velocity.Vec[2] flt64 64          "appid18,pktid1"
      448

                              7           ORBSTATETIME      Time to
propagate from                SEC
      orbital_state_time      flt64 64          "appid18,pktid1"  1728
```

Sample Command File – XML Format: cmd_import.xml

```
<database>CTDB
      <mission>DEEP IMPACT
            <cmd>
                  <object_id></object_id>

      <object_path>DI,Flyby,ADF,APPID18,PKTID1</object_path>
                  <old_cmd_name></old_cmd_name>
                  <new_cmd_name>ADFOP_INPUTS</new_cmd_name>
                  <cmd_desc>Set orbit propagation inputs</cmd_desc>
                  <cmd_type>SWC</cmd_type>
                  <cmd_bit_length>2024</cmd_bit_length>
                  <delay></delay>
                  <predecessor></predecessor>
                  <is_critical>FALSE</is_critical>
                  <cmd_constraints></cmd_constraints>
                  <tlm_response>
                    <tlm_id></tlm_id>
                    <tlm_name></tlm_name>
                  </tlm_response>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>1</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEPOSX</new_param_name>
                        <param_desc>Point to propagate from - Position
X</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KM</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[0]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>0</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>2</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEPOSY</new_param_name>
                        <param_desc>Point to propagate from - Position
Y</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KM</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[1]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>64</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>3</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEPOSZ</new_param_name>
                        <param_desc>Point to propagate from - Position
Z</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KM</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[2]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>128</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>4</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEVELX</new_param_name>
                        <param_desc>Point to propagate from - Velocity
X</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KPS</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[0]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>320</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>5</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEVELY</new_param_name>
                        <param_desc>Point to propagate from - Velocity
Y</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KPS</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[1]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>384</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>6</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATEVELZ</new_param_name>
                        <param_desc>Point to propagate from - Velocity
Z</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>KPS</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>

      <fsw_var_name>orbital_state_position.Vec[2]</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>448</pkt_start_bit>
                  </cmd_param>
                  <cmd_param>
                        <cmd_param_id></cmd_param_id>
                        <param_number>7</cmd_number>
                        <old_param_name></old_param_name>
                        <new_param_name>ORBSTATETIME</new_param_name>
                        <param_desc>Time to propagate from</param_desc>
                        <default_value></default_value>
                        <value></value>
                        <EU>SEC</EU>
                        <input_type></input_type>
                        <conv_to_DN></conv_to_DN>
                        <range></range>
                        <fsw_var_name>orbital_state_time</fsw_var_name>
                        <data_type>flt64</data_type>
                        <param_bit_length>64</param_bit_length>
                        <pkt_id></pkt_id>
                        <pkt_path>appid18,pktid1</pkt_path>
                        <pkt_start_bit>1728</pkt_start_bit>
                  </cmd_param>
            </cmd>
      </mission>
</database>
```

Sample Telemetry File – Text Format: tlm_import.txt

```
mission_id  mission     object_id   object_path tlm_id      old_name
      new_name    desc  data_type   bit_length  EU    range conv_to_EU
      flight_DN_EU_conv monitor_limits    pkt_id      pkt_path
      start_bit_in_packet

      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[0]
"                 ADFORBSTATEPOSX   Progagated Point 1 position X flt64
      64    KM          "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.0,"   FALSE             "appid18,pktid2"  112


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[1]
"                 ADFORBSTATEPOSY   Progagated Point 1 position Y flt64
      64    KM          "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.0,"   FALSE             "appid18,pktid2"  176


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_position,Vec[2]
"                 ADFORBSTATEPOSZ   Progagated Point 1 position Z flt64
      64    KM          "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.0,"   FALSE             "appid18,pktid2"  240


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[0]
"                 ADFORBSTATEVELX   Progagated Point 1 velocity X flt64
      64    KPS         "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.0"    FALSE             "appid18,pktid2"  432


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[1]
"                 ADFORBSTATEVELY   Progagated Point 1 velocity Y flt64
      64    KPS         "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.1"    FALSE             "appid18,pktid2"  496


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_velocity,Vec[2]
"                 ADFORBSTATEVELZ   Progagated Point 1 velocity Z flt64
      64    KPS         "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0,
C5=+0.2"    FALSE             "appid18,pktid2"  560


      Deep Impact
      "DI,Flyby,ADF,ADCSOP,APPID18,PKTID2,orbital_state_time"
      ADFORBSTATETIME   Progagated Point 1 Time flt64 64    SEC
      "C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0, C4=+0.0, C5=+0.0"      FALSE
            "appid18,pktid2"  752
```

Sample Telemetry File – XML Format: tlm_import.xml

```
<database>CTDB
      <mission>Deep Impact
            <mission_id>1</mission_id>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_position, Vec[0]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEPOSX</new_name>
                  <desc>Progagated Point 1 position X</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KM</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>112</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_position, Vec[1]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEPOSY</new_name>
                  <desc>Progagated Point 1 position Y</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KM</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>176</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_position, Vec[2]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEPOSZ</new_name>
                  <desc>Progagated Point 1 position Z</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KM</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>240</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_velocity, Vec[0]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEVELX</new_name>
                  <desc>Progagated Point 1 velocity X</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KPS</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>432</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_velocity, Vec[1]</object_path>
```

```
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEVELY</new_name>
                  <desc>Progagated Point 1 velocity Y</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KPS</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>496</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_velocity, Vec[2]</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATEVELZ</new_name>
                  <desc>Progagated Point 1 velocity Z</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>KPS</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>560</pkt_start_bit>
            </tlm>
            <tlm>
                  <object_id></object_id>
                  <object_path>DI, Flyby, ADF, ADCSOP, APPID18, PKTID2,
orbital_state_time</object_path>
                  <tlm_id></tlm_id>
                  <old_name></old_name>
                  <new_name>ADFORBSTATETIME</new_name>
                  <desc>Progagated Point 1 Time</desc>
                  <data_type>flt64</data_type>
                  <bit_length>64</bit_length>
                  <EU>SEC</EU>
                  <range></range>
                  <conv_to_EU>C0=+0.0, C1=+1.0, C2=+0.0, C3=+0.0,
C4=+0.0, C5=+0.0</conv_to_EU>
                  <flight_DN_EU_conv>false</flight_DN_EU_conv>
                  <monitor_limits></monitor_limits>
                  <pkt_id></pkt_id>
                  <pkt_path>appid18,pktid2</pkt_path>
                  <pkt_start_bit>752</pkt_start_bit>
            </tlm>
      </mission>
```

```
</database>
```

Appendix B

Core Classes

SCATS.java

```
package scats;

public class SCATS {
      // String holding the path to any export files
      public static String exportPath = "C:\\Documents and 
Settings\\Shea\\workspace\\scats\\temp\\export\\";
      // String holding the path to any import files
      public static String importPath = "C:\\Documents and 
Settings\\Shea\\workspace\\scats\\temp\\import\\";
      // object that used to control how the framework components 
execute
      public static SCATSController boss;


      public static String getExportPath(){
            return exportPath;
      }
      public static String getImportPath(){
            return importPath;
      }

      /**
       * @param args
       */
      public static void init(String[] args) {
            System.out.println("Initializing SCATS system, please 
standby...");

            // run database checks
            // - verify database connection; exit program if no 
database found
            // - verify core database tables are present; create if 
necessary
            // - verify only one copy of SCATS is currently running 
with specified database
            SCATSDB.init();

            // create a controller
            try {
                  boss = new SCATSController();
                  // initialize the controller
                  boss.init();
            } catch (SingletonException e) {
                  System.out.println(e.getMessage());
            }
      }
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		init(args); // initialize scats
		// test SCATS functionality
		boss.testSCATS();
	}
}
```

SCATSController.java

```
package scats;

import org.w3c.dom.Document;

public class SCATSController {
	static boolean instance_flag = false; // true if one instance
exists

	/**
	 * @param args
	 */
	public void init(){
		System.out.println("  initializing the controller
object...");
	}

	public SCATSController() throws SingletonException {
		if (instance_flag)
			throw new SingletonException("Error: Only one
SCATSController instance is allowed");
		else
			instance_flag = true;
		System.out.println("  SCATSController instantiated...");
	}

	public void testSCATS(){
		// test importing a XML object hierarchy configuration file
		FileTypeContext importContext = new FileTypeContext();
		importContext.setXMLFile();
		SCATSImport importer = new SCATSImport();
		Document doc =
importer.importCTConfig(SCATS.getImportPath() +
"object_hierarchy_config.xml", importContext);

		// test exporting a XML object hierarchy configuration file
		FileTypeContext exportContext = new FileTypeContext();
		exportContext.setXMLFile();
		SCATSExport exporter = new SCATSExport();
		exporter.exportCTConfig(SCATS.getExportPath() +
"object_hierarchy_config.xml", importContext, doc);
	} // end testSCATS()

}
```

SCATSDB.java

```
package scats;

import java.sql.*;

public class SCATSDB {
      private static String databaseName = "scats";
      private static String databaseHostName = "prosna.cbio.psu.edu";

      // get/set private variables
      public static String getDBName(){
            return databaseName;
      }
      public static void setDBName(String dbname){
            databaseName = dbname;
      }

      public static String getDBHostName(){
            return databaseHostName;
      }
      public static void setDBHostName(String dbhostname){
            databaseHostName = dbhostname;
      }

      // run database checks
      public static void init(){
            verifyDBConnection();
            verifyDBCoreTables();
            verifyUniqueSCATS();
      }

      // - verify database connection; exit program if no database
found
      public static void verifyDBConnection(){
            System.out.println("  verifying that scats can connect to
the specified database...");
            try {
                  //Get a connection to the database for a valid user
named ctdb_admin with the appropriate password
                  Connection con = DatabaseConnector.getConnection();

                  System.out.println("    Connection: " + con);

                  con.close();
            } catch( Exception e ) {
                  e.printStackTrace();
            }//end catch
      }

      // - verify core database tables are present; create if necessary
      // t_dbconfig, t_user, t_object, t_type
      public static void verifyDBCoreTables(){
            System.out.println("  verifying that the core database
tables exist...");
      }
      // - verify only one copy of SCATS is currently running with
specified database
      public static void verifyUniqueSCATS(){
            System.out.println("  verifying that only one copy of the
scats software is running with specified database...");
      }

}
```

SCATSExport.java

```
package scats;

import org.w3c.dom.Document;

public class SCATSExport {

      // export SCATS configuration file
      // instance method; ANY valid user can run this at ANY time)
      public boolean exportSCATSConfig(String filePath, FileTypeContext
ctx, Document doc){
            System.out.println("  starting export of SCATS
configuration file...");

            ctx.write(filePath, doc);

            System.out.println("  completed SCATS configuration file
export");
            return true;
      }

      // export command and telemetry object hierarchy configuration
file
      // instance method; ANY valid user can run this at ANY time)
      public boolean exportCTConfig(String filePath, FileTypeContext
ctx, Document doc){
            System.out.println("  starting export of object hierarchy
configuration file...");

            ctx.write(filePath, doc);

            System.out.println("  completed object hierarchy
configuration file export");
            return true;
      }
}
```

SCATSImport.java

```
package scats;

import org.w3c.dom.Document;
```

```
public class SCATSImport {

      // import SCATS configuration file;
      // (only ONE valid user can run this at A time)
      public boolean importDBConfig(String filePath, FileTypeContext
ctx){
            System.out.println("  starting import of SCATS
configuration file...");
            Document doc = ctx.read(filePath);
            ctx.printDocument(doc);
            // update the database object table with the imported
config file
            updateDBConfig(doc);
            System.out.println("  completed SCATS configuration file
import");
            return true;
      }

      // update the object hierarchy stored in the database
      public boolean updateDBConfig(Document doc){
            System.out.println("  updating database configuration...");
            return true;
      } // end updateDBConfig()


      // import command and telemetry object hierarchy configuration
file; class method
      // class method; only ONE valid user can run this at A time)
      public Document importCTConfig(String filePath, FileTypeContext
ctx){
            System.out.println("  starting import of object hierarchy
configuration file...");
            Document doc = ctx.read(filePath);
            ctx.printDocument(doc);
            // update the database object table with the imported
config file
            updateDBObjectHierarchy(doc);
            System.out.println("  completed object hierarchy
configuration file import");
            return doc;
      }

      // update the object hierarchy stored in the database
      public boolean updateDBObjectHierarchy(Document doc){
            System.out.println("  updating database with new object
hierarchy...");

            return true;
      } // end updateDBObjectHierarchy()
}
```

File Manipulation Classes

FileTypeContext.java

```
package scats;

import org.w3c.dom.*;

//this object is used to select a strategy when reading/writing files
based on requested file format (txt, xml, excel)
//implemented as part of a Strategy pattern
public class FileTypeContext {
      private FileTypeStrategy fileTypeStrategy;
      private String filePath = "";

      public FileTypeContext() {
            setXMLFile(); // default strategy
      }
      public FileTypeContext(String filepath) {
            filePath = filepath;
            setXMLFile(); // default strategy
      }

      // set filePath method
      public void setFilePath(String filepath){
            filePath = filepath;
      }
      // set strategy methods
      public void setTXTFile(){
            fileTypeStrategy = new TXTFileStrategy();
      }
      public void setXMLFile(){
            fileTypeStrategy = new XMLFileStrategy();
      }
      public void setExcelFile(){
            fileTypeStrategy = new ExcelFileStrategy();
      }

      // read the file into a Document object
      public Document read(){
            return fileTypeStrategy.readFile(filePath);
      }
      public Document read(String filePath){
            this.setFilePath(filePath);
            return fileTypeStrategy.readFile(filePath);
      }

      // write the Document object out to a file
      public void write(Document doc){
            fileTypeStrategy.writeFile(filePath, doc);
      }
      public void write(String filepath, Document doc){
            this.setFilePath(filepath);
            fileTypeStrategy.writeFile(filePath, doc);
      }

      // print the object hierarchy to console
      public void printDocument(Document doc){
            System.out.println ("Printing document to console:");
            // get root node of xml tree structure
            Node root = doc.getDocumentElement();
            printNode(root, 5);
        System.out.println ("Printing complete");
      }

      /** Returns element value
       * @param elem element (it is a XML tag)
       * @return Element value otherwise empty String
       */
      public final static String getElementValue( Node elem ) {
            Node kid;
            if( elem != null){
                  if (elem.hasChildNodes()){
                        for( kid = elem.getFirstChild(); kid != null;
kid = kid.getNextSibling() ){
                              if( kid.getNodeType() == Node.TEXT_NODE
){
                                    return kid.getNodeValue();
                              }
                        }
                  }
            }
            return "";
      }

      private String getIndentSpaces(int indent) {
            StringBuffer buffer = new StringBuffer();
            for (int i = 0; i < indent; i++) {
                  buffer.append(" ");
            }
            return buffer.toString();
      }

      /** Writes node and all child nodes into System.out
       * @param node The XML node (from a XML tree) from which the
output statement will start
       * @param indent The number of spaces used to indent output at
each level
       */
      public void printNode(Node node, int indent) {
            // get element name
            String nodeName = node.getNodeName();
            // get element value
            String nodeValue = getElementValue(node);
            // get attributes of element
            NamedNodeMap attributes = node.getAttributes();
            System.out.println(getIndentSpaces(indent) + "NodeName: " +
nodeName + ", NodeValue: " + nodeValue);
            for (int i = 0; i < attributes.getLength(); i++) {
                  Node attribute = attributes.item(i);
                  System.out.println(getIndentSpaces(indent + 2) +
"AttributeName: " + attribute.getNodeName() + ", attributeValue: " +
attribute.getNodeValue());
            }
            // write all child nodes recursively
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++) {
                  Node child = children.item(i);
                  if (child.getNodeType() == Node.ELEMENT_NODE) {
                        printNode(child,indent + 2);
                  }
            }
      }
}
```

FileTypeStrategy.java

```
package scats;

import org.w3c.dom.Document;

//implemented as part of a Strategy pattern
public abstract class FileTypeStrategy {

      // read/write are implemented in derived classes - one per file
format
      public abstract Document readFile(String filePath);
      public abstract boolean writeFile(String filePath, Document
objHierarchy);

}
```

ExcelFileStrategy.java

```
package scats;

import java.io.*;
import org.w3c.dom.Document;

import org.apache.poi.poifs.filesystem.*;
import org.apache.poi.hssf.usermodel.*;

//implemented as part of a Strategy pattern
public class ExcelFileStrategy extends FileTypeStrategy {

      // read object hierarchy from an Excel file and store contents in
a Document object
      public Document readFile(String filePath) {
            try {
                POIFSFileSystem fs = new POIFSFileSystem(new
FileInputStream(filePath));
                HSSFWorkbook wb = new HSSFWorkbook(fs);
                HSSFSheet sheet = wb.getSheetAt(0);
                HSSFRow row;
                HSSFCell cell;

                int rows; // No of rows
                rows = sheet.getPhysicalNumberOfRows();

                int cols = 0; // No of columns
                int tmp = 0;
                // This trick ensures that we get the data properly
even if it doesn’t start from first few rows
                for(int i = 0; i < 10 || i < rows; i++) {
                    row = sheet.getRow(i);
                    if(row != null) {
                        tmp =
sheet.getRow(i).getPhysicalNumberOfCells();
                        if(tmp > cols) cols = tmp;
                    }
                }

                for(int r = 0; r < rows; r++) {
                    row = sheet.getRow(r);
                    if(row != null) {
                        for(int c = 0; c < cols; c++) {
                            cell = row.getCell((short)c);
                            if(cell != null) {
                                // Your code here
                            }
                        }
                    }
                }
                return null;
            } catch(Exception ioe) {
                ioe.printStackTrace();
            }
            return null;
      } // end readFile()

      // write object hierarchy (stored in Document object) out to a
file in Excel format
      public boolean writeFile(String filePath, Document objHierarchy){
            try {
                  // create a new file
                  FileOutputStream out = new
FileOutputStream(filePath);
                  // create a new workbook
                  HSSFWorkbook wb = new HSSFWorkbook();
                  // create a new sheet
                  HSSFSheet s = wb.createSheet();
                  // declare a row object reference
                  HSSFRow r = null;
                  // declare a cell object reference
                  HSSFCell c = null;

                  //create 50 cells
                  for (short cellnum = (short) 0; cellnum < 50;
cellnum++) {
                      //create a blank type cell (no value)
                      c = r.createCell(cellnum);
                  }
                  // write the workbook to the output stream
                  // close our file (don't blow out our file handles
                  wb.write(out);
                  out.close();
                  return true;
            } catch (IOException e){
            }
            return false;
      } // end writeFile()
}
```

TXTFileStrategy.java

```
package scats;

import org.w3c.dom.Document;

//implemented as part of a Strategy pattern
public class TXTFileStrategy extends FileTypeStrategy {

      // read object hierarchy from an txt file and store contents in a
Document object
      public Document readFile(String filePath) {
            return null;
      } // end readFile()

      // write object hierarchy (stored in Document object) out to a
file in txt format
      public boolean writeFile(String filePath, Document objHierarchy){
            return false;
      } // end writeFile()

}
```

XMLFileStrategy.java

```
package scats;

import java.io.*;

import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

//implemented as part of a Strategy pattern
public class XMLFileStrategy extends FileTypeStrategy {

      // read object hierarchy from an XML file and store contents in a
Document object
      public Document readFile(String filePath) {
            DocumentBuilder docBuilder;
            Document doc = null;
            DocumentBuilderFactory docBuilderFactory =
DocumentBuilderFactory.newInstance();

      docBuilderFactory.setIgnoringElementContentWhitespace(true);
            try {
                  docBuilder = docBuilderFactory.newDocumentBuilder();
            } catch (ParserConfigurationException e){
                  System.out.println("Wrong parser configuration: " +
e.getMessage());
                  return null;
            }
            try {
                  doc = docBuilder.parse (new File(filePath));
            } catch (SAXParseException err) {
                  System.out.println ("** Parsing error" + ", line " +
err.getLineNumber () + ", uri " + err.getSystemId ());
                  System.out.println(" " + err.getMessage ());
                  return null;
            } catch (SAXException e) {
                  System.out.println("Wrong XML file structure: " +
e.getMessage());
                  Exception x = e.getException ();
                  ((x == null) ? e : x).printStackTrace ();
                  return null;
            } catch (IOException e){
                  System.out.println("Could not read source file: " +
e.getMessage());
            } catch (Throwable t) {
                  t.printStackTrace ();
            }
            return doc;
      } // end readFile()

      // write object hierarchy (stored in Document object) out to a
file in XML format
      public boolean writeFile(String filePath, Document doc){
            System.out.println("Saving XML file... " + filePath);
            // open output stream where XML Document will be saved
            File xmlOutputFile = new File(filePath);
            FileOutputStream fos;
            Transformer transformer;
            try {
                  fos = new FileOutputStream(xmlOutputFile);
            } catch (FileNotFoundException e) {
                  System.out.println("Error: " + e.getMessage());
                  return false;
            }
            // Use a Transformer for output
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            try {
                  transformer = transformerFactory.newTransformer();
            } catch (TransformerConfigurationException e) {
                  System.out.println("Transformer configuration error:
" + e.getMessage());
                  return false;
            }
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(fos);
            // transform source into result will do save
            try {
                  transformer.transform(source, result);
            } catch (TransformerException e) {
                  System.out.println("Error: (during transform) " +
e.getMessage());
            }
            System.out.println("XML file saved.");
            return true;
      } // end writeFile()

}
```

Utility Classes

DatabaseConnector.java

```
package scats;

import java.sql.*;

public class DatabaseConnector {
      private static Connection connection;

      public static Connection getConnection() {
            if (connection != null)
                  return connection;
            Connection con = null;
            String driver = "com.mysql.jdbc.Driver";
            try {
                  Class.forName(driver).newInstance( );
            } catch (Exception e) {
                  System.out.println("Failed to load mySQL driver.");
                  return null;
            }
            try {
                  //Define URL of database server for
                  // database named ctdb on the host computer
(prosna.cbio.psu.edu)
                  // with the default port number 3306.
                  String url = "jdbc:mysql://prosna.cbio.psu.edu/ctdb";
                  con = DriverManager.getConnection(url, "ctdb_admin",
"ctdb_a(dm1N)0");
            } catch (Exception e) {
                  e.printStackTrace( );
            }
            connection = con;
            return con;
      }

}
```

SingletonException.java

```
package scats;

public class SingletonException extends RuntimeException {
      public SingletonException() {
            super();
      }
      public SingletonException(String s){
            super(s);
      }
}
```