

Automated Commanding of the SMAP Spacecraft Enables Efficient, Reliable, and Responsive Operations

Christopher Swan
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-4133
Christopher.A.Swan@jpl.nasa.gov

Abstract- The Soil Moisture Active Passive (SMAP) mission developed and deployed a system to autonomously handle most routine commanding of the observatory. This system of ground software is able to build commands, validate them, and radiate the commands to the spacecraft, all without human interaction. In the case of an off-nominal scenario, the system will abort gracefully and notify the mission operations team of the problem. The system was phased into operations during the first three months of the SMAP mission and handles over 90% of the weekly commanding of the vehicle. The gradual introduction of the automation in flight, along with an extensive test campaign, was instrumental in the success of the software. The automation has enabled substantial efficiencies in operations team staffing and has improved reliability by removing the potential for human error. The system also allows the SMAP project to be more responsive which has shown significant benefits in areas of data latency and science accuracy.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. SMAP AUTOMATION ARCHITECTURE	1
3. HERITAGE TO TEST SCRIPTING.....	1
4. AUTOMATION DESIGN: REQUIREMENTS	2
5. AUTOMATION DESIGN: IMPLEMENTED	4
5. TEST PROGRAM.....	6
6. FLIGHT PERFORMANCE	6
7. BEYOND SMAP.....	7
8. CONCLUSION.....	8
ACKNOWLEDGEMENTS	8
REFERENCES.....	8
BIOGRAPHY	8

1. INTRODUCTION

The Soil Moisture Active Passive (SMAP) mission is a NASA Earth satellite mission that provides global surface soil moisture measurements on a frequent basis [1]. The observatory is equipped with two instruments, radar¹ and a radiometer, which share a spinning six meter antenna. The spinning antenna allows for a measurement swath width of

¹ On July 7th, 2015 SMAP experienced an anomaly with the radar’s high-power amplifier and was unable to recover it [2]. The mission continues to operate and produce valuable data using the radiometer and the anomaly did not impact the routine operations of the spacecraft.

1000km which allows SMAP to make complete soil moisture maps of the Earth every 2-3 days. This data is downlinked via frequent communication opportunities with the NASA Near Earth Network (NEN) and operated out of the Jet Propulsion Laboratory (JPL).

2. SMAP AUTOMATION ARCHITECTURE

The automation of various aspects of operating a spacecraft is not a new concept for JPL [3][5]. The advances that the SMAP project has implemented relate to the level and scope of automation. To put this into context, it is useful to categorize the three different levels of automation: DWN, UP, CMD. (Figure 1)

Downlink (DWN) automation primarily acts to process and transfer data throughout the ground data system. Once the ground data is available, the Uplink (UP) automation can use it to build command products for uplink. The capability SMAP has implemented goes one step further with the creation of Commands (CMD), the ability to command observatory without a flight controller on console. Most JPL missions utilize a combination of systems that perform some amount of the “DWN” and “UP” level automation; the “CMD” automation is a new development in autonomous ground software and the focus of this paper.

3. HERITAGE TO TEST SCRIPTING

The SMAP approach to in-flight CMD automation was based, in large part, on experience with automated testing methods used in the ground test campaigns of JPL spacecraft. Both the Flight Software and the Flight Systems team utilized Python based test scripts, which allows for efficient regression and tabulation of test results. This is enabled by a Python library which interfaces to both the command and the telemetry portions of the JPL ground data system. Using the library, test engineers are able to send commands to and query telemetry from both the hardware testbed and the spacecraft simulator. Over multiple JPL Missions, additional libraries have been developed that build on that basic capability. These libraries incorporate features such as automated telemetry checking for every command sent, or the capability to record and report the pass and fail status of requirements within the script. Unlike the initial library, which is multi-mission, these augmented libraries are adapted and modified for a given project.

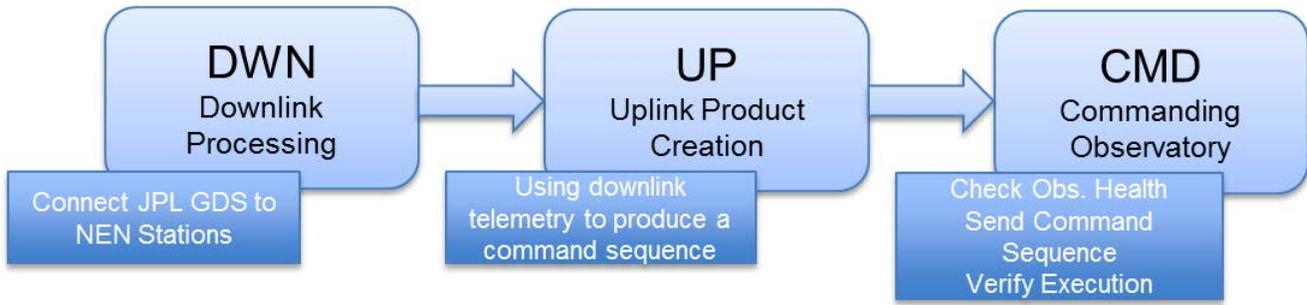


Figure 1 the different levels of automation on SMAP.

This experience and institutional knowledge was not lost during the development of the SMAP Automation infrastructure and was directly referenced during early requirements discussions. Following the testing approach, the JPL Ground System initially delivered a set of basic interface functions, which could apply to any mission, in a Python library. This provided the capability to interface to the command and telemetry systems of a spacecraft inflight. The SMAP Flight Operations team, led by the Science Phase Lead Christopher Swan, then augmented these basic functions in a project library that was designed to support the specific SMAP mission requirements.

4. AUTOMATION DESIGN: REQUIREMENTS

The goal of the SMAP automation effort was to automate routine commanding to the vehicle to increase reliability and efficiency of the operations team. In order to ensure reliability, the design of the SMAP project automation library (SMAP AUTO) was deeply rooted in a core JPL principle “Test as you Fly, Fly as you Test”. With that in mind, the core design of SMAP AUTO was largely constrained by what could be effectively tested. This was decomposed into a set of design requirements which were applied across the library:

- Abort and notify operations team in off-nominal scenarios

Responding intelligently to off-nominal scenarios can quickly become extremely complicated and drive both development and test complexity. As SMAP AUTO was intended for routine commanding this was not necessary, and simply aborting the execution of the session and notifying the operation team is sufficient.

- Automation sessions are self-contained and are not reliant on prior actions

Maintaining knowledge on the status of other ground systems and prior automation sessions could allow for more efficient actions but would drive complexity. Instead SMAP AUTO opted to limit external interfaces and design each automation session to be self-contained.

- Automation sessions cannot run in parallel with other automation or manual commanding sessions.

Maintaining the knowledge of the actions of ground operators and other automation sessions would be a major complexity driver. SMAP AUTO is limited to a single automation session at a time and is not designed to be run in parallel with manual operations.

- Automation code must not change from test to flight venues
- Enable in-flight testing without radiation of products

While SMAP AUTO was extensively tested using the SMAP observatory testbed (hardware in the loop) it was not possible to fully simulate all the operational systems or environments in the test venue. It was also necessary to force off-nominal conditions in the test venue that would not be possible in the flight venue. This drove a software design that was configurable by venue, without requiring a code change, and could be fully executed against the spacecraft in flight to the point of generating commands, without sending them (for on-orbit testing).

Once these design requirements were established, the functional requirements and interfaces were established. The primary interface to the rest of the ground data system is via the AMPCS (AMMOS Mission Data Processing and Control System) AUTO interface allows SMAP AUTO to access real time telemetry and radiate commands to the SMAP observatory. An additional interface manages the SMAP ground system and connecting to the various Near Earth Network (NEN) stations. This application, the Pass Automation Daemon (PAD), initiates the SMAP AUTO automation sessions at the beginning of specified communication passes and provides an interface for SMAP AUTO to query pass information (e.g. time left in the pass)[4].

The individual automation functions are identified by decomposing the actions the flight operations team performs to execute each activity [5]. These actions are augmented

with additional verification steps to ensure any off-nominal scenario is detected. An example of this decomposition and how it mirrors the manual process is shown in Table 1.

Table 1 SMAP ephemeris activity decomposed by function

Manual Process	SMAP AUTO step	Functional Decomposition
Process starts at least 24-48 hours before uplink	Process starts at the beginning of the uplink pass.	
Engineer locates latest ephemeris	SMAP AUTO locates the latest delivered ephemeris	smap_find_bsp()
Engineer queries telemetry to locate currently executing on board ephemeris	SMAP AUTO queries telemetry for currently executing on board ephemeris	Smap_cheby_check()
Engineer builds binary files and command products Ephemeris command product is a “template” or sequence block SMAP manual process generates this via script (or UP Automation)	SMAP AUTO builds binary files and command products.	Smap_build_cheby() Smap_make_ephem_seq()
Engineer tests products	SMAP AUTO tests new ephemeris file against currently executing ephemeris file	Smap_cheby_diff()
Other engineers review products	Not required due to extensive testing and review of SMAP AUTO codebase.	
Command approval meeting held		
Time scheduled to uplink products (minimum two engineers required)		
Uplink established, engineers review telemetry	SMAP AUTO waits for uplink lock, then checks spacecraft health, time remaining in the pass	Smap_go_for_command() Smap_health_check() Smap_pass_time_left()
Binary products radiated and receipt confirmed.	SMAP AUTO then radiates each product <ul style="list-style-type: none"> • Prior to radiating each product SMAP AUTO reconfirms spacecraft health, uplink lock, sufficient time remaining • Telemetry is collected before and after each uplink and SMAP AUTO compares them to confirm uplink receipt. • Uses 5-10 telemetry types depending on the product² • Uplinks are ordered so that the command to start loading the binary products occurs last. 	Smap_bulk_scmf_upload()
Command products radiated and receipt confirmed. Typically based on one or two pieces of telemetry		Smap_send_fsw_cmd() Smap_health_check() Smap_pass_time_left() Smap_uplink_telem()
Engineer delivers ephemeris products and logs to archive location	SMAP AUTO then delivers ephemeris products and logs to archive location	

² A human operator would not reconfirm basic telemetry between each uplink or check so many telemetry channels because they have a deeper understanding of what they are seeing. These are examples of areas where telemetry checks were augmented to ensure off-nominal scenarios were detected.

5. AUTOMATION DESIGN: IMPLEMENTED

The implementation of SMAP AUTO evolved into three separate pieces: A library of functions for performing each activity (`smap_auto.py`), a set of templates which group the functions together to perform a given activity, and a configuration file used for fine tuning performance and adapting to changes in execution venue.

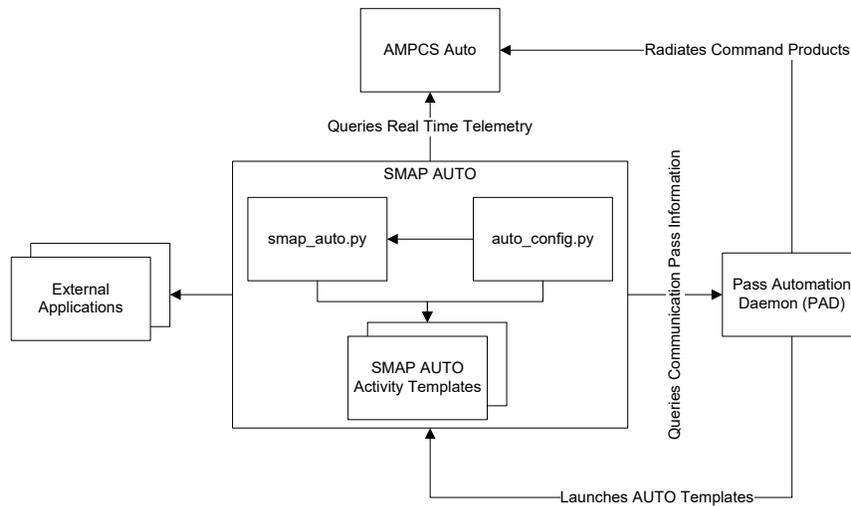


Figure 2 SMAP AUTO design

The SMAP AUTO functions query telemetry via the AMPCS AUTO interface and sends commands by placing command products in a protected directory that PAD monitors and then radiates (also using the AMPCS AUTO interface). SMAP AUTO activities are launched by PAD at the beginning of a communication pass based on activity templates files placed in a monitored directory and named according to the communication pass in question. In certain instances, the functionality required in SMAP AUTO is performed by other applications in the ground data system. Rather than reinvent that capability, SMAP AUTO utilizes the external applications but validates or tests the output to detect any off-nominal results. During development several external applications were modified in order to provide more explicit feedback to SMAP AUTO regarding off-nominal conditions.

In order to meet the design requirements regarding testing the SMAP AUTO library and activity templates, which contain all logic and functions to perform automation, does not change from test to flight venues. The configuration file, which is limited to defining a set of variables, is allowed to change to accommodate both changes in venue and operation performance tuning. To accommodate inflight testing, any activity template (and the underlying library functions) that autonomously builds commands products has

a “no uplink” variant that produces all the commands, checks all the telemetry, but stops short of radiating them to the spacecraft.

As of October 2015, SMAP has sixteen defined activity templates which all follow the same pattern for implementation:

- Confirm that only a single instance of SMAP AUTO is running
- Monitor spacecraft telemetry until S-Band receiver reports uplink carrier and bit lock
- Check spacecraft telemetry to verify the health of the spacecraft
- Check spacecraft telemetry to determine the uplink data rate detected by the S-Band receiver
- Execute the activity
- Check spacecraft telemetry to verify the health of the spacecraft

Each action is implemented by one or more functions in the SMAP AUTO library depending on their complexity. These functions fall into either a generic category used by all activities or activity specific functions designed to support a single activity.

Table 2 SMAP AUTO functions by category

Category	Number of Functions	Description of functions
Generic	20	Logging and Performance Monitoring Query of Real Time Telemetry Query of Recorded Telemetry Build Command products Send Command products (individually or a list) Telemetry Checks (health, uplink rate, command receipt)
Activity Master	3	Queries currently executing command sequences Finds current trajectory Executes external application to build command sequence with routine maintenance commanding
Bad Block	7	Builds memory mask file Queries memory address pointers and determines if masking memory block would corrupt onboard science data (aborts if true) Builds command sequence to apply memory mask file after communication pass is complete (cannot apply mask while downlinking)
Ephemeris	8	Finds current trajectory Executes external application to build onboard ephemeris file Checks for discontinuity between new and executing ephemeris files Builds command sequence to apply ephemeris file
Instrument Look Up Table (LUT) Synchronization	4	Finds current trajectory Executes external application to produce sync command sequence based on current trajectory
Command Loss	1	Updates and logs command loss timer
Data Management	5	Executes external application to produce list of products to delete or retransmit Build command sequence(s) to perform actions
Science Retransmit	7	Executes external application which tracks gaps in science data Builds commands to execute retransmission

Table 3 SMAP AUTO activity templates

Activity Template	Description
smap_auto_lut_sync.py smap_auto_lut_sync_no_uplink.py	Updates onboard instrument commanding (based on reference trajectory) account for actual trajectory
smap_auto_activity_master.py smap_auto_activity_master_no_uplink.py	Command sequence that performs routine maintenance of the spacecraft and schedules preplanned activities
smap_auto_data_mgmt.py smap_auto_data_mgmt_no_uplink.py	Deletes and retransmits engineering products based on what has been received on the ground
smap_auto_ephem.py smap_auto_ephem_no_uplink.py	Updates onboard trajectory based on ground prediction
smap_auto_sci_rexmit.py smap_auto_sci_rexmit_no_uplink.py	Commands retransmission of science data based on gaps identified
smap_auto_bad_block_mask.py smap_auto_bad_block_mask_no_uplink.py	Masks the use of blocks of memory in a manner that does not corrupt existing data
smap_auto_file_upload.py	Uplinks a list of approved, manually produced command products.
smap_auto_set_clt.py	Updates the command loss timer
smap_auto_no_op.py	Sends a "NO OP" Command (for testing)
smap_auto_health_check.py	Checks the spacecraft telemetry for spacecraft health (for testing)

5. TEST PROGRAM

In order to ensure that the SMAP observatory could be entrusted to the SMAP AUTO infrastructure for the majority of its regular commanding, a rigorous test program was put into place. It made extensive use of the SMAP Hardware Test Bed and nearly all the pieces of the ground data system (in a test venue). The testing started at the function level and proceeded to test entire activity templates in both nominal and off-nominal scenarios. The off-nominal testing included both spacecraft and ground centric failures and tested all possible logical paths in the automation code. Once this was complete, the entire SMAP AUTO infrastructure was exercised in an extended mission scenario test spanning multiple days of continuous simulated flight (mirroring the state of the flying SMAP spacecraft).

While this proved to be an effective test venue, it still lacked fidelity when compared to the actual spacecraft and ground system in-flight. As a result, the next step in the test program was to execute SMAP AUTO activities in “shadow mode”, where they were executed in the flight venue and allowed to perform all their functions except for command uplink. The flight operations team monitored the results,

manually tested the output products, and then radiated them at later time. This on-orbit testing typically lasted several weeks to a month depending on the complexity of products built. The results of these tests uncovered issues and idiosyncrasies with the operational system unable to be detected in the test venue. These ranged from software launch application issues (system service vs. command line), to telecommunications behavior (coherent transmit causes a momentary drop in bit-lock when the frequency shifts to the forward link signal), to timing of telemetry checks. The shadow mode testing proved to be invaluable and drove changes to the automation code and configuration, as well as other elements of the ground system software. After the shadow mode tests were completed successfully, the SMAP AUTO capability was allowed to command without a human operator in the loop.

6. FLIGHT PERFORMANCE

The SMAP AUTO capability was checked out and phased in during the first three months of operations (the Commissioning Phase) and was fully operational by the start of the Science phase of the mission (April 20th, 2015).

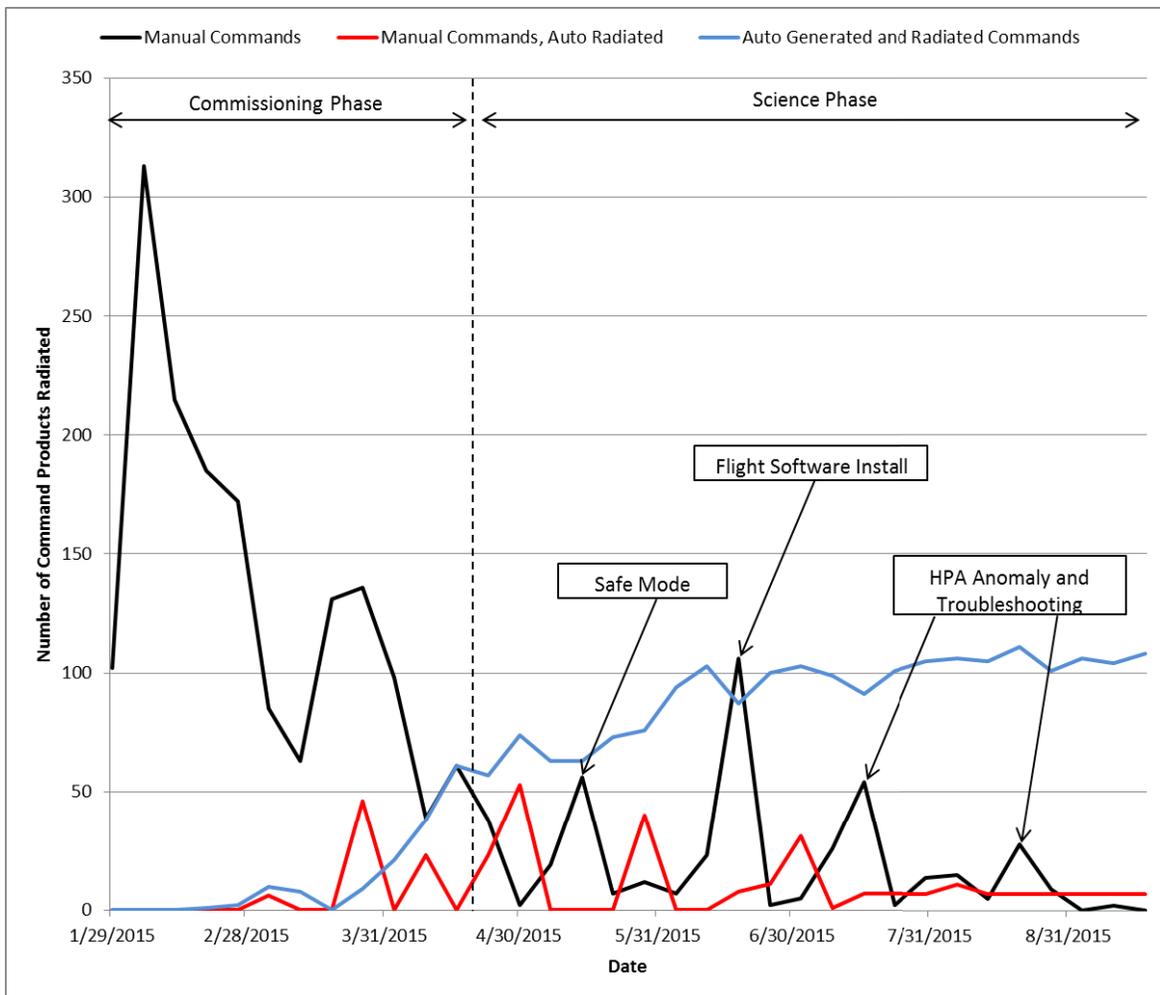


Figure 3 In-Flight Manual vs. Automated Commanding

As shown in Figure 3 the use of SMAP AUTO allowed manual commanding of the spacecraft to drop rapidly, to below 10% of the total commanding, increasing only to support non-routine operations (anomalies and a flight software installation). In addition to the fully automated commanding capability of SMAP AUTO, the SMAP Flight Operations team used it to radiate manually built and tested products (via `smap_auto_file_upload.py`). This was especially useful for transmitting large binary files (e.g. Instrument Look up Tables), which would take multiple communication passes to complete. The SMAP AUTO infrastructure allowed the flight operations team to queue the files for overnight radiation without needing operations staff on console. Once the non-routine operations had passed, the manual commanding dropped below 10% of total commanding.

Table 4 Percent manual vs. automated commanding

	Manual Commands	Manual Commands, Auto Radiated	Auto Generated and Radiated Commands
Start of Science Phase	15% (394)	8% (218)	76% (1984)
August, 2015 – September, 2015	7% (59)	6% (53)	87% (752)

During the first nine months of operations, SMAP AUTO detected off-nominal scenarios and aborted its activity a total of seventy six times. The primary source of these scenarios was related to problems detected in the ground software or the source files that SMAP AUTO used to build command products. Issues related to station outages from offline Near Earth Network (NEN) stations or delayed initial lock up with the spacecraft and the NEN sites were also detected (as a failure to receive initial telemetry). SMAP AUTO also detected problems via the interaction with the spacecraft. These were primarily related to command products that failed to be successfully uplinked and successful command receipt could not be confirmed. In all cases, AUTO detected the anomaly, aborted all actions and notified the flight operations team via email and SMS messaging.

Table 5 Off-Nominal sources of SMAP Automation aborts

Ground Software	43
Station/Communication Outage	16
Spacecraft Detected	17
Total	76

The use of SMAP AUTO allowed the SMAP operations team to become both more flexible and more responsive to the requirements driving the routine operations. Since SMAP AUTO runs without a human in the loop, increased usage does not impact staffing plans - so the functions could be run much more frequently than specified in the original operations plan (Table 6). This improved performance in the

areas of data latency via more frequent retransmissions. Also attitude control from daily updates (vs. semi-weekly) provided an order of magnitude improvement in pointing accuracy. SMAP AUTO also made operational retransmission of science data possible. While the spacecraft and ground software supported science retransmission, the flight operations team could not support it because of the timeframe (12-24 hours) before the onboard data was overwritten. SMAP AUTO provided the capability to respond to missing science data with 4-6 hours, enabling science data retransmission. The increase in frequency of the activities also provided the flight operations team with flexibility and extra margin against operations requirements. The failure of any individual activity to complete is not in itself an issue, as SMAP AUTO will execute it again, well before violating any operations requirements. If a failed activity does need to be executed immediately, the SMAP AUTO infrastructure allows rapid scheduling of a “make-up” activity in the required time frame.

The use of SMAP AUTO enhanced operational efficiency and reduced the effort needed to manually operate the SMAP spacecraft. Considering the man-hours needed to manually perform the routine actions at the required frequency, the SMAP AUTO realized a savings of approximately one fulltime engineer of effort. This would be dramatically higher, considering the workforce required to manually command the vehicle at the frequency that was actually implemented. In order to meet the current (September 2015) cadence of commanding the SMAP operation team would need to expand from a regular workweek schedule (5/40) to a constant presence on console (24/7). This would require at least 8-10 additional full time engineers. SMAP AUTO was able to effortlessly scale the frequency of commanding with no additional workforce.

Table 6 Implemented vs. Required Frequency of Automated Activities

Automation Activity	Required Frequency	Implemented Frequency
Data Management	Weekly	6hrs
Command Loss	Semi-weekly	Daily
Instrument Command Table Sync	Semi-weekly	Daily
Activity Master	Weekly	Weekly
Ephemeris Update	Semi-weekly	Daily
Bad Block Mask	Weekly	Weekly
Science Retransmission	Every 4-6 hours	Every 4-6 hours

7. BEYOND SMAP

As with test scripting infrastructure it was based on, the SMAP AUTO capability is specific to the SMAP mission, but the underlying approach and interfaces are applicable to all missions. The capability of interfacing with the command and telemetry system would remain but details regarding which commands are sent and which telemetry items are checked would change from mission to mission. Other

differences could include communication pass duration, one way light time, external applications used, and the specific routine activities for that mission. These differences also exist in the manual processes and procedures from mission to mission. Once the activities are identified and the processes are outlined they can be considered for automation.

8. CONCLUSION

The SMAP AUTO infrastructure is an effective method to take care of routine flight operations in a reliable and responsive way. The combination of focused requirements and robust test campaign, including an on orbit test program, resulted in a reliable system trusted to safely perform a discrete and routine set of operations. This capability has demonstrated, in-flight, measurable improvements in the areas of mission operations efficiency, reliability, and performance of data latency/loss and attitude control. While the SMAP automation infrastructure was explicitly designed for the SMAP operations scenario, the same methods could be applied to other missions.

ACKNOWLEDGEMENTS

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

I would like to acknowledge Antonio Sanders, the developer of PAD, for all his help in supporting the design and test of SMAP AUTO.

I would like to acknowledge Eric Rigor, Lavin Zhang, Rob Puncel, Duane Morgan, Reynaldo Lopez-Roig, Stuart Gibson, and Bob Wing for all their help support in testing SMAP AUTO.

Lastly, I would like to acknowledge Tracy Drain and Lisa Swan for their excellent feedback and critic of this paper.

REFERENCES

- [1] SMAP Project Website. Jet Propulsion Laboratory, n.d. Web. 12 Dec. 2015. <<http://smap.jpl.nasa.gov/>>.
- [2] "NASA Soil Moisture Radar Ends Operations, Mission Science Continues." SMAP Project Website. N.p., 2 Sept. 2015. Web. 14 Dec. 2015. <<http://smap.jpl.nasa.gov/news/1247/>>.
- [3] Pack, M., and S. Laubach (2014), "Evolution of the scope and capabilities of uplink support software for mars surface operations", paper presented at 13th International Conference on Space Operations, SpaceOps 2014, May 5, 2014 - May 9, 2014, Pasadena, CA, American Institute of Aeronautics and Astronautics Inc.
- [4] Sanders, A. (2014). "Automating the SMAP Ground Data System to Support Lights-Out Operations", paper presented at 13th International Conference on Space Operations, SpaceOps 2014, May 5, 2014 - May 9, 2014, Pasadena, CA, American Institute of Aeronautics and Astronautics Inc.
- [5] (in publication) Tung, Ramona (2016). "Development of Effective and Efficient Operations for NASA's Soil Moisture Active Passive Mission", 37th IEEE Aerospace Conference 2016.

BIOGRAPHY



Christopher Swan received a B.S. in Aerospace Engineering from Illinois Institute of Technology in 2002 and a M.S. in Astronautical Engineering from USC in 2007. He is currently the Science Phase Lead for the SMAP project. Previously he worked as flight systems engineer for the SMAP spacecraft with a focus on system operability. Prior to SMAP, he was a surface and payload system engineer on the Phoenix Mars mission, supporting both System I&T and surface operations.