Automated Operations Development for Advanced Exploration Systems

Angie T. Haddock [1]
angie.haddock@nasa.gov
NASA, Marshall Space Flight Center, Huntsville, AL 35812, US

Howard K. Stetson [2]
howard.k.stetson@nasa.gov
Teledyne Brown Engineering/NASA, Marshall Space Flight Center, Huntsville, AL 35812, US

Abstract—**Automated space operations command and control software development and its implementation must be an integral part of the vehicle design effort. The software design must encompass autonomous fault detection, isolation, recovery capabilities and also provide "single button" intelligent functions for the crew. Development, operations and safety approval experience with the *Timeliner* system on-board the International Space Station (ISS), which provided autonomous monitoring with response and single command functionality of payload systems, can be built upon for future automated operations as the ISS Payload effort was the first and only autonomous command and control system to be in continuous execution (6 years), 24 hours a day, 7 days a week within a crewed spacecraft environment. Utilizing proven capabilities from the ISS Higher Active Logic (HAL) System [1] , along with the execution component design from within the HAL 9000 Space Operating System [2] , this design paper will detail the initial HAL System software architecture and interfaces as applied to NASA's Habitat Demonstration Unit (HDU) in support of the Advanced Exploration Systems, Autonomous Mission Operations project. The development and implementation of integrated simulators within this development effort will also be detailed and is the first step in verifying the HAL 9000 Integrated Test-Bed Component [2] designs' effectiveness. This design paper will conclude with a summary of the current development status and future development goals as it pertains to automated command and control for the HDU.**

I. Introduction

In April 2012, the International Space Station (ISS) laboratory module Destiny will have been supporting science research in Earth orbit for over eleven years. Many of the spacecraft development and scientific research accomplishments of the ISS program have been documented in the media, professional journals and academic publications. One of the vitally important systems supporting ISS is a software tool commonly referred to as "Timeliner."

During the early years of ISS operations, NASA-Marshall Space Flight Center (MSFC) ISS Payload Operations and Integration Center (POIC) Flight controllers were routinely sending dozens of commands daily just to configure data systems, activate and deactivate experiments and many other routine, complex, yet necessary activities in order to successfully utilize the ISS as a world-class science facility.

Fortunately, NASA had anticipated that this would be the case and had planned to use the Timeliner User Interface Language (UIL) on ISS. The Timeliner UIL [3] was developed by the Charles Stark Draper Laboratory in 1981 for use in simulating tasks performed by astronauts aboard the Space Shuttle. In 1992, Timeliner was selected by NASA as the user interface language for the ISS, and it was incorporated into the ISS Command and Control Multiplexer-DeMultiplexer (MDM) and the Payload MDM (PLMDM).

Beginning in 1999, software engineers at the POIC, working with Draper, began developing autonomous blocks of software, called Timeliner Bundles in new and innovative ways to reduce ground controller workload, add reliability and to a certain extent, put a virtual-controller onboard. From those humble beginnings to the present, engineers, scientists and flight controllers have developed an autonomous and continuously executing system onboard the ISS called Higher Active Logic (HAL). The HAL System performs autonomous monitoring with

[1] Computer Engineer, Space Systems Operations/Mission Operations Lab, NASA, Marshall Space Flight Center.
[2] Computer Scientist, Space Systems Operations/Mission Operations Lab, TBE/NASA, Marshall Space Flight Center.

command responses for Payload Systems and Subsystems. The design of the ISS HAL System was the proof of concept for the execution component for the larger HAL 9000 System design.

In September 2011, software engineers at MSFC were asked to begin development of incorporating Timeliner-TLX™ within the NASA's Habitat Demonstration Unit/Deep Space Hab (HDU/DSH) (Figure 1.) simulation environment in support of the Advanced Exploration Systems (AES), Autonomous Mission Operations (AMO) project, and convert Procedure Representation Language (PRL) [4] scripts into Timeliner-TLX™ scripts. This paper describes the development and analysis of integrated simulators within the Timeliner-TLX™ environment, how to "qualify" the Timeliner-TLX™ scripts from the converted PRL scripts, testing all paths of execution, and how the initial HAL System software architecture and interfaces are applied to HDU/DSU. The development and implementation of integrated simulators within this development effort will also be detailed and is the first step in verifying the HAL 9000 Integrated Test-Bed Component [2] designs' effectiveness. This design paper will conclude with a summary of future development goals as it pertains to automated command and control for deep space missions.

Figure 1: NASA's Habitat Demonstration Unit/Deep Space Hab

II. Procedure Execution Capability

Once the ability to convert PRL procedures to Timeliner-TLX™ auto-procedures was accomplished, a rapid way was needed to execute the procedures for analysis and comparison of logic flows. The ability to insert faults within the command paths and at each system/sub-system was also desired in order to fully exercise the procedure by traversing all paths of execution. The HAL 9000 System design included an integrated test bed with resident simulations (Figure 2.) for verification and validation of newly created procedures. Could the simulation be embedded within the Timeliner-TLX™ Execution Components or would they be required to be external simulations? The decision was made to implement an integrated simulation within Timeliner-TLX™ that would respond to the commands within the converted PRL HDU/DSH procedures.

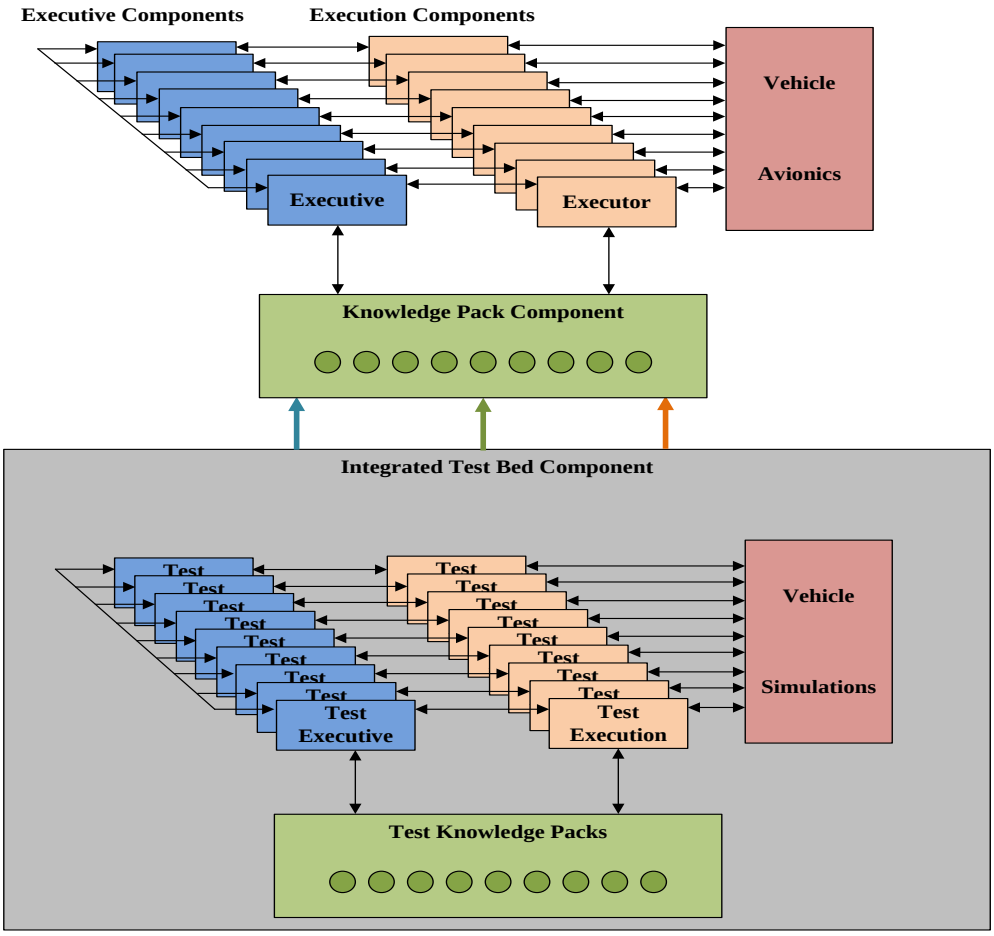

Figure 2: HAL 9000 Context Diagram

This opened the door for initial design and development of Timeliner-TLX™ based simulations for auto-procedure interaction that could be employed within the HAL9000 Test Bed. The possibility of up-linking the simulation and converted PRL procedures for scenario testing with ISS directed that the auto-procedures and simulation be compatible with the Timeliner Executor resident on the International Space Station Payload Multiplexer / De-Multiplexer (PLMDM).

III. HAL9000 Simulation Integration Context

The HAL 9000 Integrated Test Bed (Figure 3) includes the 9 planning engines, the 9 Timeliner-TLX™ Execution engines, and the 9 Timeliner-TLX™ simulators (4 of which utilized for HDU/DSH simulation). The Test Bed would be utilized for qualifying crew authored, in-flight developed procedures during the mission. These procedures would be required to pass test bed execution in order to be hosted in the real-time environment. The initial development and checkout of internal Timeliner-TLX™ simulations for HDU/DSH PRL procedure conversion support and the data exchanges that would be required has shown a limited but valuable aspect to procedure qualifying in that all paths of execution can be induced to verify the logic of the procedure and its flow control when anomalies are encountered, but does not utilize a true command interface.

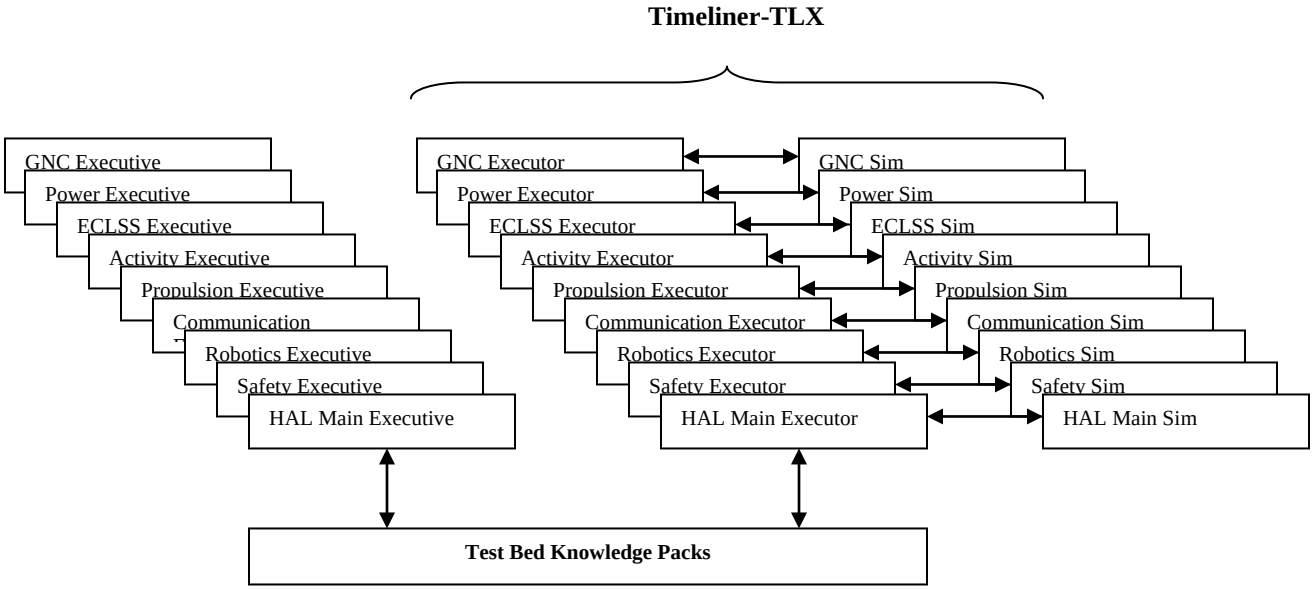

Figure 3: HAL 9000 Test Bed Context

IV. Command Handling

Since the procedures and simulation both execute within the same Timeliner-TLX™ Execution Engine, the capability to format and transmit a command to the simulation without using the Timeliner-TLX™ "Command" statement was needed. Timeliner-TLX™ Command Statements are linked directly to external I/O interfaces and will be used to send commands to actual HDU/DSH Hardware. A conversion option was needed to place command data into memory that would be monitored by the simulation. The ISS HAL System 3 [1] instituted a ground request handling function that essentially had the capability required. The PRL converter was designed with an option that would place all command data and routing into Timeliner Current Value Table (CVT) variables. These variables reside within global memory available to all Timeliner-TLX™ auto-procedures. The HDU/DSH Timeliner-TLX™ simulation would monitor for this data, interpret the routing and command data and perform the end item data generation that was utilized by the executing procedure. The converter also has the option of generating the Timeliner-TLX™ Command statements when desired to send commands to the actual HDU/DSH.

The Simulation was divided into separate command handlers in an attempt to mirror the HAL9000 Execution Component design, the Power handler, the Communications handler, the Environmental Control and Life Support System (ECLSS) handler, and the Activity handler. The HAL 9000 system design also includes Propulsion, Guidance Navigation and Control (GNC), Safety, Robotics and HALMain functional separations but these were deemed not to be required for the HDU/DSH procedure execution at this time.

The Power command handler performs only functions related to the HDU/DSH power system and sub-systems. The ECLSS command handler performs only functions related to HDU/DSH ECLSS and Thermal Control Systems (TCS). In the HAL9000 design TCS is part of ECLSS and not classified separately. The COMM command handler performs only functions related to communication systems and subsystems as well video systems. The Activity command handler performs all "activity" related functions such as avionics configurations, and utility type functions such as turning lights on/off. This division of command functions follows the planning and execution functions of the HAL 9000 System design and also allows a division of processing within the simulation to not only assist in negating a memory race condition, but also to allow multiple commands to be processed simultaneously when commanding different systems and sub-systems when multiple procedures are in execution. Nominal procedure logic qualification will be one at a time, but some procedures invoke procedure steps within other procedures which drives the requirement for simultaneous procedure execution.

Upon development and initial testing of the command handlers with the initial converted PRL procedures, an execution "race" condition was apparent in that the execution timing within the Timeliner-TLX™ execution engine

allowed a converted PRL procedure to attempt multiple commands in its "time slice" before the targeted command handler was able to read memory in its "time-slice". Although the number of statements per execution pass is configurable within the Timeliner-TLX™ system, it was decided to use "Wait" statements after each command in order to synchronize the command requests from the PRL procedures to the HDU/DSH simulation. Issuing a Wait statement within a Timeliner-TLX™ auto-procedure causes the procedure to "give up" its' time slice, effectively allowing the simulation the time to process the command data request. This condition arises due to the nature of the procedure authorship where the author issues several commands without checking the end items until all commands are sent. This is a "rapid fire" command method or command chain which was unexpected as nominally end items are checked after each command is sent and before sending the next command in the procedure. The time period to wait was settled at 3 seconds, 1 second to release the time slice, 1 second to process the data and perform the end-item function, and 1 second to message the console of the function performed. An example of this type of procedure was encountered in the initial PRL procedures where power is commanded to a Power Distribution Unit (PDU), and the commands to power other devices interfaced to the PDU are sent immediately thereafter. If the PDU did not power up or required a power up time period, all other commands to that PDU would be rejected and the procedure author would not know of the failure until after execution as the end items of the commands were not checked until after all commands were sent. The HDU/DSH Timeliner-TLX™ Simulation command interface context is shown in Figure 4.

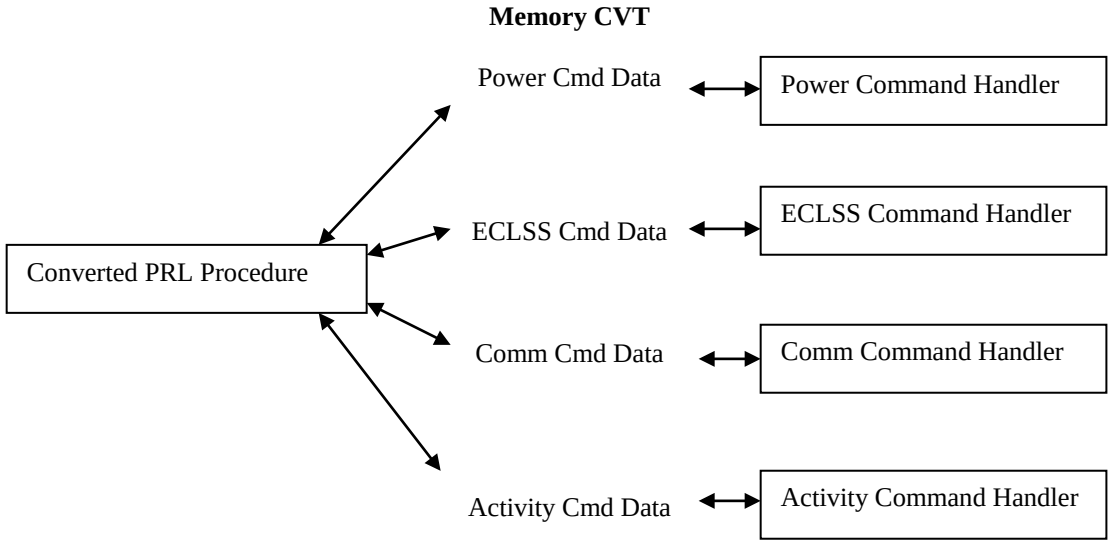

Figure 4: HDU/DSH Command Interface Context

In the HAL 9000 System design, the request handlers require a "Sequence ID" be provided by the requesting sequence for validating the source of the request. This is an important function for negating code and procedure errors during execution but with PRL procedures and their conversion, there was no way to assign unique sequence id's since upon conversion each PRL "step" was converted to a sequence, each PRL procedure was converted individually, the order and number of PRL procedures can change and the HAL 9000 Development Methodology [2] was not employed as in this methodology all procedures are Timeliner-TLX™ procedures from beginning of operations development and are assigned unique identifiers.

V. Command Handler Code

The sample code segment in Figure 5 depicts a 1 second control loop where the request handler samples memory for a command request. The handler only looks for a command id that is not zero, and if one arrives, flags that a request is in progress. Requesting sequences must check to see if a request is in progress and if the command id is equal to zero before populating a request. This will make a sequence wait until a request has completed before issuing another request. Note also the code for identifying the sequence making the request. As mentioned earlier, this source validation has not been implemented, but the code remains.

```
Set ACTYRequestInProgress = 0   -- none in progress at this point
every 1.0         -- Look for a command every second
   if HDUACTYRequestID /= 0 then  -- do we have a request?
   set ACTYRequestInProgress = 1  -- we got a request, indicate processing
   set RequestComplete = 0       -- Indicte not completed
   set RequestingSeqID = HDUACTYRequestingSeqID -- who sent us the request?
   set CurrentRequest = HDUACTYRequestID - 100 -- save the request ID
```

Figure 5: Sample Control Loop Code

The sample code segment in Figure 6 depicts a function execution within the Activity Command Handler of the HDU/DSH Simulation.

```
-- *************************
-- *** Check for XHAB      ***
-- *************************
if SystemID = 4 then -- XHAB
   if SubSys1ID = 14 then -- CRIO
      if SubSys2ID = 16 then  -- CTRL1
         set XHAB_CRIO_CTRL1_OPERATION_MODE_SENSOR_1 = HDUACTYRequestData1
         Message "ACTY: XHAB CRIO CTRL1 OPERATION MODE SENSOR 1 SET"
      end if
   end if
   if SubSys1ID = 16 then -- HUMFAC
      if SubSys2ID = 16 then  -- CTRL1
        set XHAB_HUMFAC_CTRL1_OPERATION_MODE_SENSOR_1 = HDUACTYRequestData1
        Message "ACTY: XHAB HUMFAC CTRL1 OPERATION MODE SENSOR 1 SET"
      end if
   end if
   if SubSys1ID = 17 then -- INFLATION
     if SubSys2ID = 16 then  -- CTRL1
       set XHAB_INFLATION_CTRL1_OPERATION_MODE_SENSOR_1 = HDUACTYRequestData1
       Message "ACTY: XHAB INFLATION CTRL1 OPERATION MODE SENSOR 1 SET"
     end if
   end if
end if
```

Figure 6: Function Code Sample

The command routing data was read from memory as well as the command id. The routing data is checked , in the first case XHAB->CRIO->CTRL1. The code structure allows the insertion of a fault flags anywhere along the path and including the end item data. An example of fault insertion is depicted in Figure 7.

```
-- *************************
-- *** Check for XHAB      ***
-- *************************
if SystemID = 4  and XHABFault = False then -- XHAB
   if SubSys1ID = 14 and CRIOFault = False then -- CRIO
      if SubSys2ID = 16 and CTRL1Fault = False then  -- CTRL1
         set XHAB_CRIO_CTRL1_OPERATION_MODE_SENSOR_1 = HDUACTYRequestData1
         Message "ACTY: XHAB CRIO CTRL1 OPERATION MODE SENSOR 1 SET"
      end if
   end if
end if
```

Figure 7: Command Routing Code Sample

The routing information within the command data is used to determine the logical and or physical path the command travels before reaching the destination. The end item variable(s) are set in memory, a message is generated to trace the execution and the requesting procedure can now check the end items for continued execution or anomaly resolution.

VI. Fault Insertion

The ability to insert a non-response capability to a command or to insert a failure signature (data indication) is extremely important in the auto-procedure certification process. During the initial ISS auto-procedure development, it was required by the MSFC Mission Operations Laboratory (MOL), the ISS Computer Safety Working Group, the Payload Software Control Panel (PSCP), the Timeliner Operations Review Panel (TORP) and the Avionics Software and Control Board (ASCB), that all paths of execution be tested. This was accomplished during rigorous testing with the Space Station Training Facility (SSTF) and the Payload Software Integration and Test Facility (PSIVF), by utilization of numerous data load commands to on-board memory setting up the data conditions prior to execution. This method encompassed many hours of test time as well as producing and documenting the test results proving all paths of execution were employed during certification testing. Each auto-procedure had to be executed numerous times in order to explore each path of execution. The creation of auto-procedures for path and logic control of auto-procedures being certified allows real-time insertion of faults during the testing, negating numerous commanding prior to test execution. It also reduces the test time as well as the amount of facility resources needed for such path and logic testing. It does not fully qualify an auto-procedure though as actual flight hardware or flight equivalent units or flight certified simulations must be utilized for the final qualification for flight readiness. It should also be noted that during ISS testing with external facilities, it became very difficult to data load failures since these were joint Operational Readiness Tests (ORT), not all procedure paths could be induced due to impacts to other test groups participating in the ORT. The internal simulation capability alleviates some of these problems.

VII. Conclusion

The design described in this paper enhances the development of automated operations for the AES-AMO. The Timeliner-TLX™ language and system is a vital requirement, as without its execution structure and language components, the design could not be accomplished or, at a minimum, the increased costs and development schedule impact would adversely affect our progression. The Timeliner-TLX™ system is flight qualified and human rated allowing for immediate development use. Future development will support deep space missions by allowing the crew single command functionality for on-board functions, crew authoring of procedures, and crew validation/verification of procedures on-board. This will allow the crew to develop and execute procedures on-board and will reduce communication requirements with earth-based assets for procedure execution in case of communication delays and/or permanent loss of communications. The “pre-qualification” of crew authored auto-procedures can be done via the integrated Timeliner-TLX™ simulations. This pre-qualification provides for logic path testing of all paths and fault insertions. High fidelity testing will be required to fully qualify an auto-procedure for real-time execution, as external simulations/services will be required in order to declare a procedure as “flight ready”. Currently, the future goals are the development of a diagnostic failure model of an HDU/DSH subsystem, and to automate the model by interfacing with real-time telemetry, providing failure data to Timeliner-TLX™ Command & Control (C&C) systems and having Timeliner-TLX™ procedures perform the responses.

AMO’s mission is to demonstrate advanced capabilities for autonomous and crew-centered operations, with emphasis on reducing the crew’s workload and dependence on mission support from Earth-based control centers. This design paper describes a small effort within AMO’s mission and will further the advancement toward on-board procedure authorship and validation/verification.

Appendix A
Acronym List

AES	Advanced Exploration Systems
AMO	Autonomous Mission Operations
ASCP	Avionics Software and Control Panel
C&C	Command & Control
COMM	Communications
DSH	Deep Space Hab
ECLSS	Environmental Control and Life Support System
GbE	Gigabyte
GNC	Guidance , Navigation and Control
HAL	Higher Active Logic
HDU	Habitat Demonstration Unit
HDU/DSH	Habitat Demonstration Unit/Deep Space Hab
ISS	International Space Station
ISD	Integrated Server Device
MDM	Multiplexer/Demultiplexer
MOL	Mission Operations Lab
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
ORT	Operational Readiness Tests
PDU	Power Distribution Unit
PLMDM	Payload Multiplexer/Demultiplexer
POIC	Payload Operations and Integration Center
PRL	Procedure Representation Language
PSCP	Payload Software Control Panel
PSIVF	Payload Software Integration and Test Facility
SSMMU	Solid State Mass Memory Unit
SSTF	Space Station Training Facility
TBE	Teledyne Brown Engineering
TCS	Thermal Control Systems
TLX	Timeliner Execution Environment
TORP	Timeliner Operations Review Panel
UIL	User Interface Language
VCC	Vital Communication Computer

Appendix B
Glossary

HAL 9000 Space Operating System	HAL 9000 Space Operating System is a crew-integrated, autonomous command and control system designed specifically for fully automated, long-duration deep space vehicles.
Habitat Development Unit/Deep Space Hab	Habitat Development Unit-Deep Space Hab is a one story, 4-port habitat unit with an approximate volume of 56 cubic meters. The HDU/DSH shell can accommodate an inflatable loft for additional laboratory or habitation volume. HDU/DSH project is a multi-center team project consisting of NASA architects, scientist, and engineers, working together to develop sustainable living quarters, workspaces, and labs for next-generation space missions.

Acknowledgments

The authors would like to acknowledge Craig Cruzen, Mission Operations Laboratory, NASA Marshall Space Flight Center, Scott Akridge, Space Systems Department, NASA Marshall Space Flight Center, Shen Chang, Spacecraft & Vehicle Systems Department, NASA Marshall Space Flight Center, and Dr. Jeremy Frank, NASA Ames Research Center, for their support with the initial groundwork of incorporating Timeliner- TLX™ into NASA's Habitat Demonstration Unit/Deep Space Hab simulation environment and PRL scripts to Timeliner-TLX™ scripts conversion work in support of the Advanced Exploration Systems, Autonomous Mission Operations.

References

[1] Stetson, H. K.; Deitsch, D. K.; Cruzen, C. A., Haddock, A. T.; "Autonomous Operations Onboard the International Space Station," IEEE Aerospace Conference, Big Sky, Montana, March 2007.

[2] Stetson, H. K.; Knickerbocker, G. K.; Cruzen, C. A., Haddock, A. T.; "The HAL 9000 Space Operating System," IEEE Aerospace Conference, Big Sky, Montana, March 2011.

[3] Brown, R. A.; Braunstein, R.; Brunet, R. C.; Grace, R.; Vu, T.; Busa, J.; Dwyer, W. K.; "Timeliner: Automating Procedures on the ISS," 2002 SpaceOps Conference, Houston, Texas, October 2002.

[4] Dalal, K. M.; Frank, J.; "Bridging the Gap between Human and Automated Procedure Execution," IEEE Aerospace, Big Sky, Montana, March 2010.

Biography

Angie T. Haddock *is employed by NASA's Marshall Space Flight Center (MSFC) in Huntsville, Alabama, in the Mission Operations Lab. Currently, Ms. Haddock is the Co-Lead of the MSFC Advanced Exploration Systems-Autonomous Mission Operations (AES-AMO), and an analyst for the Space Launch Systems (SLS) Flight Operations. In the SLS position, she has worked with the analysis and development of the SLS Flight Operations Specification and supported the analysis and documentation of requirements and Launch Commit Criteria derived from the vehicle system. Preceding the SLS and the AES projects, Ms. Haddock was the Operations Lead for the ISS PLMDM. In this role, she was responsible for leading the development of PLMDM operations, technical coordination for the development and update of PLMDM software. Prior to joining NASA in 2000, she worked for Teledyne Brown Engineering, where she trained to serve as a Command and Payload MDM Officer (CPO) for the ISS Payload Operations Integration Center. She holds a Bachelor of Science degree in Computer Science from Athens State University. Ms. Haddock, and her husband, Stacey, have three daughters; Stephanie, Mary Elizabeth & Amy.*

Howard K. Stetson *is a contractor for Marshall Space Flight Center, Space Systems Operations and is currently working as an analyst for the Space Launch Systems (SLS) flight operations, avionics and software, as well as the Advanced Exploration Systems-Autonomous Mission Operations project and has over 34 years of experience in software development and engineering. Preceding the SLS and AES projects, Mr. Stetson designed, developed and implemented the Higher Active Logic (HAL) autonomous system for ISS payloads. Mr. Stetson, an employee of Teledyne Brown Engineering, has received the NASA Space Flight Awareness Honoree Award, the Astronauts Personal Achievement Award (Silver Snoopy), and the NASA Exceptional Pubic Service Medal. Mr. Stetson is a member of the United States Selective Service System and the United States Parachute Association and currently has over 3000 jumps.*