Remote Software Application and Display Development

Brandon Sanders
Kennedy Space Center
Major: Physics
KSC OSSI Spring
18 July 2014

Remote Software Application and Display Development

Brandon Sanders[1]
Berry College, Mt. Berry, GA, 30149

The era of the shuttle program has come to an end, but only to give rise to newer and more exciting projects. Now is the time of the Orion spacecraft, a work of art designed to exceed all previous endeavors of man. NASA is exiting the time of exploration and is entering a new period, a period of pioneering. With this new mission, many of NASA's organizations must undergo a great deal of change and development to support the Orion missions. The Spaceport Command and Control System (SCCS) is the new system that will provide NASA the ability to launch rockets into orbit and thus control Orion and other spacecraft as the goal of populating Mars becomes ever increasingly tangible. Since the previous control system, Launch Processing System (LPS), was primarily designed to launch the shuttles, SCCS was needed as Kennedy Space Center (KSC) reorganized to a multiuser spaceport for commercial flights, providing a more versatile control over rockets. Within SCCS, is the Launch Control System (LCS), which is the remote software behind the command and monitoring of flight and ground system hardware. This internship at KSC has involved two main components in LCS, including Remote Software Application and Display development. The display environment provides a graphical user interface for an operator to view and see if any cautions are raised, while the remote applications are the backbone that communicate with hardware, and then relay the data back to the displays. These elements go hand in hand as they provide monitoring and control over hardware and software alike from the safety of the Launch Control Center. The remote software applications are written in Application Control Language (ACL), which must undergo unit testing to ensure data integrity. This paper describes both the implementation and writing of unit tests in ACL code for remote software applications, as well as the building of remote displays to be used in the Launch Control Center (LCC).

Nomenclature

ACL	=	Application Control Language
API	=	Application Programming Interface; specifying routines, data structures, and object classes used in a computer programming language.
ASF	=	Application Services and Framework
Compiler	=	a program that turns source code into machine code
COTS	=	Commercial Off The Shelf product
CPU	=	Central Processing Unit
CUI	=	Compact Unique Identifier
DDE	=	Desktop Development Environment
Debug	=	Detect and remove errors from software code
Function	=	Sequence of program instructions that perform a specific task
GUI	=	Graphical User Interface
IDE	=	Integrated Developing Environment
ILOA	=	Integrated Launch Operations Applications
Integer	=	mathematical integral data type
KSC	=	Kennedy Space Center
LCC	=	Launch Control Center
LCS	=	Launch Control Systems
Parameter	=	Data provided as input to a method call
PLC	=	Programmable Logic Controller

[1] ILOA Remote Software Developer Intern, NE-C1, KSC, Berry College.

Remote	=	Controlling hardware from a distance
SCCS	=	Spaceport Command and Control System
Script	=	Program that can interpret and automate the execution of a task.
Source Code	=	Collection of computer instructions written in a human-readable computer language
Stream	=	AccuRev code base configuration
String	=	Finite sequence of characters similar to a sentence
VM	=	Virtual Machine
Workspace	=	Directory where developer's files are stored in reference to AccuRev

I. Introduction

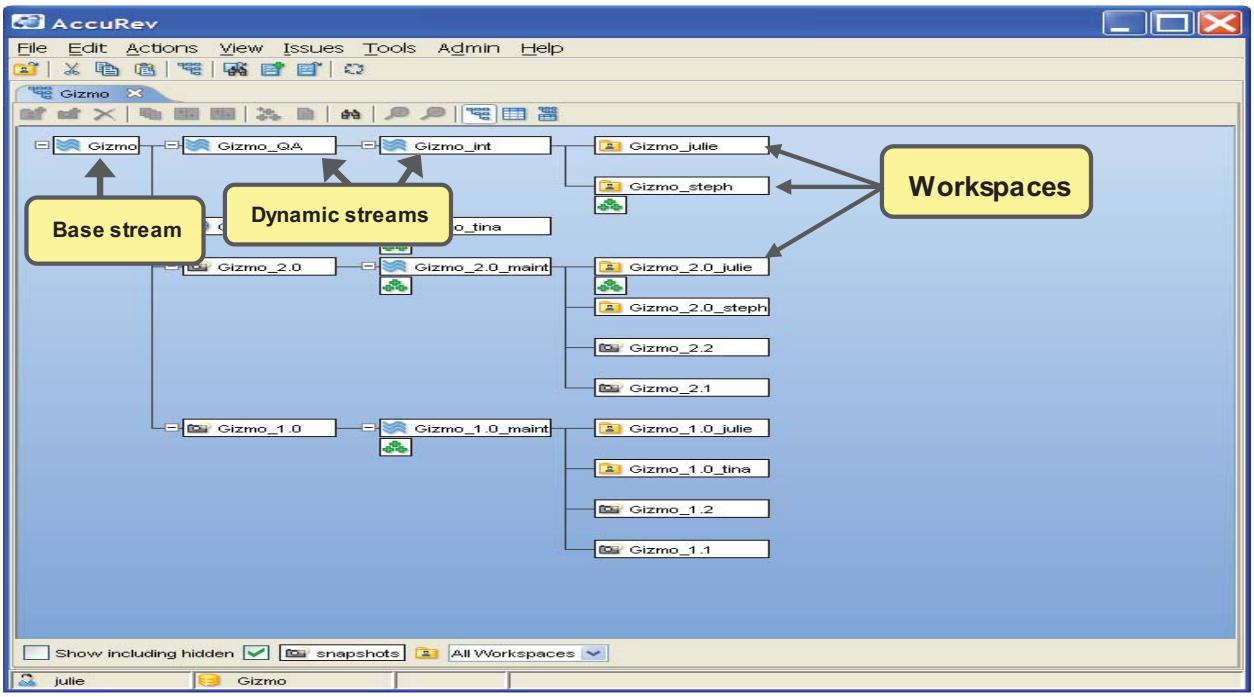
Kennedy Space Center lies at the forefront to NASA's pioneering mission of the universe. Being a ground systems and operations base, KSC is the final step before spacecraft such as Orion will be launched into space, seeking out destinations far beyond our planet. Since KSC has been in the process of reorganizing into a multi-commercial user spaceport, many changes have occurred over the past few years. Since the Shuttle's operations system known as the Launch Processing System (LPS) was highly specialized to the shuttle flights, it needed to be completely redesigned in order to support the launches of future rockets. Spaceport Command and Control System (SCCS) is the new system that has taken the place of LPS. Within SCCS, the Launch Control System (LCS) is the system in control of all of the remote software that communicates with ground and flight system hardware from a safe distance. LCS has even more subsidiaries including the Application Services and Framework (ASF) team that developed source code to translate data from flight and ground system hardware. The ASF team is also responsible for the safety checking of Application Control Language (ACL), a scripting language used by NASA engineers to monitor the rocket and ground subsystem components active during a launch. ACL is a language built by NASA engineers as an abstract layer on top of the more widely known language of C++ created by Bjarne Stroustrup. Because it is written in house per say, ACL must undergo extensive unit testing to ensure it performs as it is expected to, in order to reduce all possible risk. Unit tests are required to be run for all of ACL's API calls so there is no uncertainty to the functionality of the source code. Currently, unit tests are being developed using ACL, and are therefore helping debug the remaining problems in the code. This paper will describe the process of unit testing and provide examples of how those tests are being written and performed. Along with unit testing ACL this summer, this internship has also involved the design of remote displays under the Integrated Launch Operations Applications (ILOA) team. These displays use ACL applications to command and monitor flight and ground system hardware from the safety of the Launch Control Center (LCC). On top of ACL unit testing, this paper will also touch on what goes into the development of a remote display and the new and exciting benefits of displays today compared to those used under the shuttle program.

II. Unit Testing in ACL

A. Tools Necessary for Unit Testing

As mentioned, the ASF team is tasked with the important responsibility of unit testing every single one of ACL's API calls. The testing related to this internship involves a few different tools in order to fully complete a single unit test. First, the test is developed in Netbeans, an

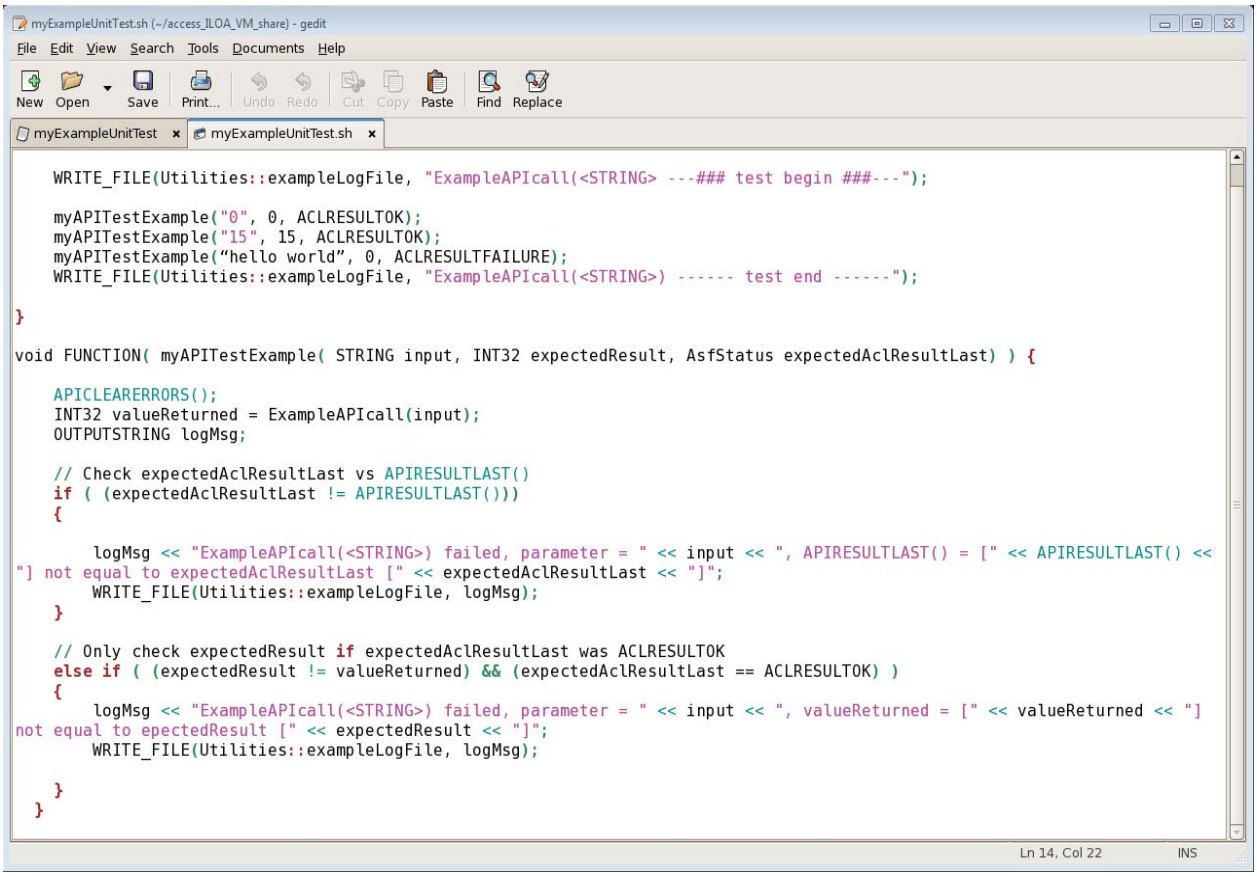
integrated developing environment (IDE) that is an application able to assist with software development. Netbeans is a commercial off the shelf product (COTS), and is configured with a custom plug-in to support the ACL API calls. After Netbeans compiles the source code successfully, the external file existing only in the user's workspace must be promoted into AccuRev. AccuRev is also COTS software and is used by KSC as a tool to optimize parallel development and preserve data integrity. As software is being written, it can be promoted in AccuRev which then sends it up into a stream. A stream is a particular configuration of the code base and helps manage the divergence and merging of the code. The stream structure is hierarchal, and once files are promoted up, all lower streams can inherit the higher level change. This allows developers to automatically merge their code effectively and receive any changes other developers introduce, reducing the time necessary for communication and meetings. Below is a graphical depiction of how AccuRev operates with the workspaces and streams pointed out.



Both Netbeans and AccuRev are accessed through a Linux Virtual Machine (VM). This allows for the Linux operating system to be running on a physical computer with another operating system, in this case Microsoft. Lastly, the finalized test must be run on a desktop development environment (DDE). This is a virtual environment that allows for the test script to be initialized and ran as it would be on a physical machine in the LCC. If no failures are logged then all is well with the test, and the ACL API call is performing as intended.

B. Writing Unit Tests, the Nitty Gritty

The most important aspect of unit testing is, of course, writing the actual test. The developer must understand what the API call he is testing is intended to do, and thus test against its performance to see if there are any unexpected results. An example unit test is described below.



First, a script is written that will execute the commands within it. Within this script, the first thing to do is write to a file that will display the outputted data. A string is written to the file stating the test has begun. In the example above the API call is ExampleAPIcall(<STRING>) with a string as its parameter and it is intended to convert a string to a 32-bit integer. A function named myAPITestExample, with parameters (STRING input, INT32 expectedResult, and AsfStatus expectedAclResultLast), is written below the script that actually supplies inputs to ExampleAPIcall(<STRING>) and based on the output, will write to an output string of any errors. The "input" parameter is the supplied string that the API call requires as an input. The "expectedResult" is the expected output from the call and the "expectedAclResultLast" parameter is an enumeration of what the call is returning based on the input. ACLRESULTOK means no errors while ACLRESULTFAILURE means an error happened during the call. If the expected result does not match the actual result, then a message will be written to the output string that will be displayed on the file. This test should pass because the expected results will be what the API method outputs. It will succeed with converting "0" to 0 and "15" to 15 but will not be able to convert "hello world" so it therefore will result in ACLRESULTFAILURE which is expected. These tedious tests are a necessary evil and will allow the developer to narrow down any bugs within the ACL API which they can then recode and fix.

C. A Necessary Evil

Unit testing takes a great deal of time and must be done correctly, because if the tests are not written as they should, then corrupt data may be displayed confusing the developer and allowing

for actual bugs to not be discovered. It can be monotonous work because API's can consist of many hundreds of method calls, but it is much more desirable to encounter a bug while testing than while writing an application that will control electronics on the rocket. Software malfunctions must be found and fixed before they can lead to any unwanted dangers. Since it is kind of a last stop before the code communicates with the hardware, ASF must ensure that their translations are working properly and sending the right commands to the end items. This shows the importance of unit testing when finalizing a software product or, in this case, the programming language used by NASA engineers to command and monitor flight and ground systems hardware and software.

III. Display Development

A. Drawing on the Display Editor

The second opportunity provided by this internship is remote software display development. These displays are created and then supplied to the LCC to be used when the launch of a rocket occurs. They monitor data of all types coming from flight and ground hardware and software. The data is linked to the display fields through a compact unique identifier (CUI). This is a sixteen digit identifier that correlates with a specific type of data coming from ground or flight software or hardware used in the launching of a rocket. In order to display the information provided by these CUI's, certain symbols must be drawn on the editor. They consist of the following:

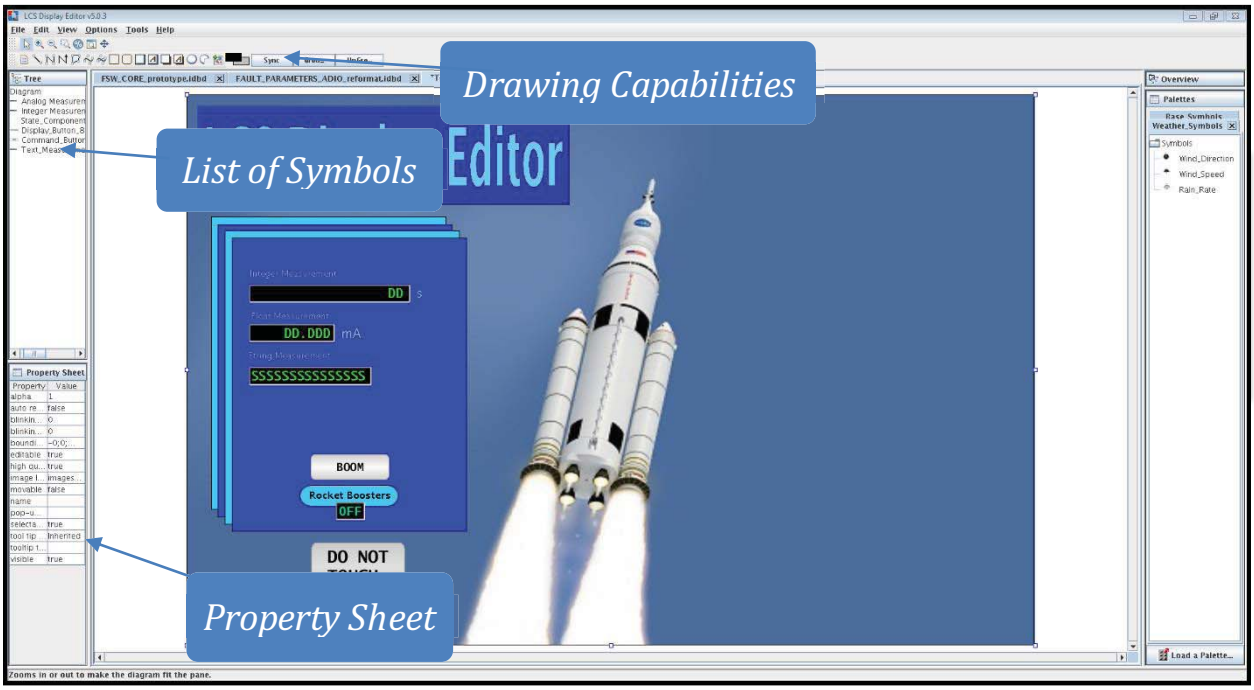
Text Measurement: Displays a measurement CUI's alphanumeric value.

Command Button: Issues a command CUI (including command parameters as required).

State Component: Displays an enumerated CUI value as a specified image.

Display Button: Calls another ILOA display

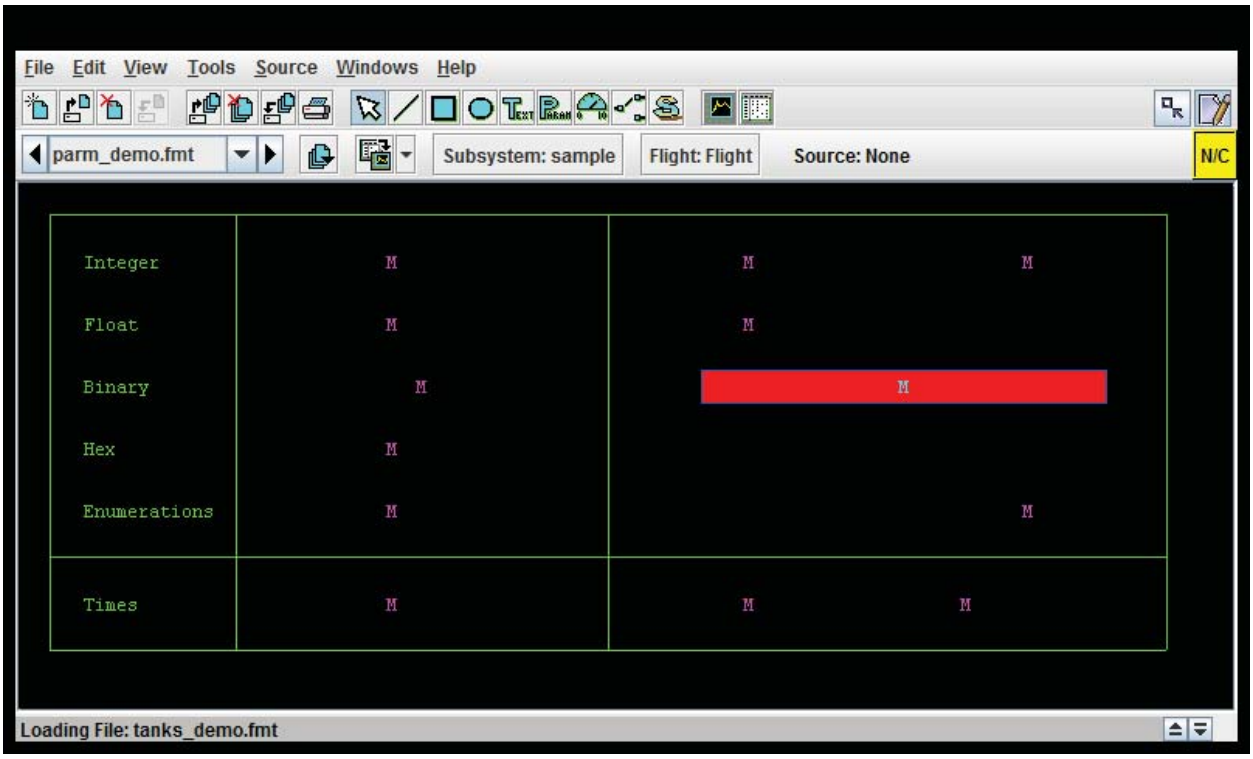
Each symbol is highly customizable and can be edited to fit the needs of the user through a properties sheet that shows the editing capability of each symbol. Below is a sample project that illustrates some of the capabilities of the LCS Display Editor.



As one can see, the displays that can be developed are highly graphical and can closely match the hardware they resemble. This allows for engineers to more easily become acquainted with the display because it looks like what they already understand. Obviously, not all displays are overly intricate due to the fact that there is a lot of information to be shown with only a few monitors available to be displayed on. Engineers must make the decisions on how their display should work and what will best represent the hardware behind the display.

B. An Obvious Upgrade

The main difference with the LCS display editor is its high graphics capabilities. Also engineers are able to write their own control applications to issue commands to ground and flight hardware. This use of current technology is a great advantage over the displays existing during the years of the shuttle program. Many of those displays were handled with the GOAL programming language. These displays were character-based graphics and needed to be controlled from the actual control applications in GOAL. A similar example to how these might have looked is given below.



As KSC becomes a multiuser spaceport, it is necessary for upgrades to be performed to keep up with the coming times, and therefore simple text based screens are no longer sufficient. With these upgrades, the LCS will be the key software system to control space launches for many years to come, incorporating new and commercially used technology that is customized to the needs of KSC.

IV. Conclusion

Kennedy Space Center is headed towards exciting times, full of innovation. Hardware is being upgraded and improved as rocket capabilities become more and more extravagant so it is simply necessary for the controlling software to be renovated and vastly improved as well. Software developers and engineers alike love their job at NASA, where they come to be creative and design new and fascinating displays and applications that will send the human race further into space. One does not land a man on Mars through sheer estimations and guesses. It takes years of well thought out plans and calculations involving thousands of engineers working side by side conquering multiple problems that arise. The transition to SCCS and LCS is not an easy one, but is indeed a necessary one that requires everyone to be on board, whether it is the ASF team closely translating ACL code into readable instructions for the ground and flight system hardware, or the ILOA team developing intricate displays that will bring in the new era of space exploration, or rather space pioneering, as NASA would like to call it, the age where humans establish a greater presence in the universe.

Acknowledgments

The author would like to thank Kurt Leucht for his mentorship as well as Linda Crawford and the rest of the ASF team who were a pleasure to work with. A thank you also goes out to Kennedy Space Center for this internship opportunity.

References

"AccuRev Basic User Training (LCS)." 2008. Document. 17 July 2014.

"LCS DSF Display Editor and Display Test Driver." 18 November 2013. Document. 17 July 2014.

Sanders, Brandon. "TrainingDisplay." n.d. Display. 17 July 2014.