Orion Scripted Interface Generator (OrionSIG)

Robert J. Dooling

NASA's Kennedy Space Center

Major: Computer Science

Program: ACCESS Summer 2013

Date: 29 July 2013

Orion Scripted Interface Generator (OrionSIG)

Robert J. Dooling[*†] (robert.j.dooling@nasa.gov)
Rochester Institute of Technology, Rochester, NY 14623

Abstract

The Orion spacecraft undergoing development at NASA and Lockheed Martin aims to launch the first humans to set foot on asteroids and Mars.[1] Sensors onboard Orion must transmit back to Earth astronomical amounts of data recording almost everything in 50,231 lb. (22,784 kg)[2] of spacecraft, down to the temperatures, voltages, or torsions of even the most minor components. This report introduces the new Orion Scripted Interface Generator (OrionSIG) software created by summer 2013 NASA interns Robert Dooling and Samuel Harris. OrionSIG receives a list of Orion variables and produces a script to graph these measurements regardless of their size or type. The program also accepts many other input options to manipulate displays, such as limits on the graph's range or commands to graph different values in a reverse sawtooth wave. OrionSIG paves the way for monitoring stations on Earth to process, display, and test Orion data much more efficiently, a helpful asset in preparation for Orion's first test mission in 2014.

Figure 1. Orion spacecraft design (*image courtesy of NASA*)

[*] Intern, NE-A2 (NASA Engineering-Avionics: Guidance, Navigation & Flight Controls), Kennedy Space Center, Rochester Institute of Technology.
[†] Superscript symbols such as "*" refer to footnotes and superscript numbers such as "[1]" refer to endnotes.

Nomenclature

CAIDA	=	Customer Avionics Interface, Development & Analysis
CUI	=	Compact Unique Identifier
CSV	=	Comma-Separated Values spreadsheet
Double	=	Decimal data type, which may store decimals up to 128 bits long
ENUM	=	Enumerator data type, which defines a constant (for example, 3.14 could define "PI")
EFT-1	=	Orion Exploration Flight Test 1
Float	=	Decimal data type, which may store decimals up to 64 bits long
FSW	=	Flight Software
GUI	=	Graphical User Interface
KSC	=	Kennedy Space Center
MPCV	=	Multi-Purpose Crew Vehicle, the function of the Orion spacecraft
NASA	=	National Aeronautics and Space Administration
OrionSIG	=	Orion Scripted Interface Generator
SI	=	Signed integer data type, which may be positive or negative
SSI	=	The filename extension of each SuperScript file
SSL	=	Honeywell Aerospace's proprietary SuperScript Language, which is the format of OrionSIG output
SQL	=	Structured Query Language
UI	=	Unsigned integer data type, which is always positive
USRP	=	Undergraduate Student Research Program

I. Introduction and Project Domain

No human has set foot on the moon since December 1972[3] and American flags planted by Apollo missions have long since faded to white sheets[4] saluting a deserted void. All of the billions of trillions of other celestial bodies in the universe, even nearby asteroids, remain completely untouched by astronauts.* The Orion Multi-Purpose Crew Vehicle (MPCV) may soon propel humans beyond low Earth orbit for the first time in over 40 years and lay the foundation for thrilling advances in space exploration. NASA's upcoming Exploration Flight Test One (EFT-1) mission scheduled for September 2014 will test the blossoming Orion spacecraft un-crewed in a launch from Cape Canaveral atop a Delta IV Heavy rocket, two orbits around the Earth, re-entry, and splashdown. On EFT-1, the Orion capsule will soar to altitudes and speeds unmatched by any spacecraft built for astronauts since Apollo 17 rocketed home from the moon four decades ago.[5]

Figure 2. Orion EFT-1 mission overview (*image courtesy of NASA*)

* Of course, un-crewed spacecraft such as the Voyager and Pioneer probes (all now well beyond Pluto) have explored much more, but the most promising part of Orion's potential lies in what humans can do that robots cannot.

A. Defining Orion Data

We faced the exciting challenge of displaying diverse types of EFT-1 data in graphs to support pre-launch testing and further Orion software development. Existing EFT-1 Flight Software (FSW) currently uses databases of Compact Unique Identifiers (CUIs). CUIs consist of strings of letters and numbers representing encoded names for variables on the Orion flight that are transmitted back to Earth during the mission. For example, the CUI OCAVXXX0XXX000XX* may refer to the current temperature in part of the crew module. In the CUI database, a temperature variable like the one in OCAVXXX0XXX000XX may be stored as a "float" (decimal) data type with a size of 64 bits.†

Figure 3. During EFT-1, Orion's capsule will re-enter Earth's atmosphere at 20,000 mph, a speed unsurpassed since 1972, and accurate data recording will help verify future astronaut safety (*image courtesy of NASA*).

Our OrionSIG solution needed to accept lists of these CUI names from the user, match them to information from a CUI database, and then generate SuperScript Language (SSL)‡ code to graph each CUI. The CUI database always exists in the same folder as OrionSIG with fields for each CUI's bus ID, start bit, number of bits, and data type.

- Bus IDs refer to the non-encoded variable names themselves.
- Start bits and numbers of bits describe where the value is located in bits.
- Finally, the data type helps us translate from binary-encoded values to actual values and vice versa.§

Retrieving the number of bits and the data type were necessary for OrionSIG to determine each CUI's maximum possible value. For example, an 8-bit unsigned integer with all bits set to 1 results in 255 and a 16-bit signed integer with all bits set to 1 results in 32767. We cannot rely on pre-defined limits because many different data types and values outside of the powers of 2, such as 11-bit integers, occur in the CUI database.

B. Monitoring Orion Data

We tested our OrionSIG project frequently in the KSC Customer Avionics Interface, Development & Analysis (CAIDA) lab. This facility and many others at NASA employ Dewesoft, a "data acquisition, test, and measurement"[6] software used for tracking and recording data during pre-flight testing, launches, and missions.** Dewesoft already enables engineers to display Orion measurement values using the bus IDs from the CUI database mentioned earlier. To complete testing of Orion data, however, OrionSIG is needed to display the full range of possible values for a CUI in all forms of graphs, from pure sine waves to square waves and reverse sawtooth waves.

Figure 4. CAIDA Logo (*image courtesy of NASA*)

Recall that OrionSIG produces graphs through SuperScript code. CAIDA includes a SuperScript Language Interface program that executes SuperScript (.ssi) files and the affected variables are plotted in Dewesoft graphs once per second. Before OrionSIG, engineers needed to manually write .ssi scripts to test any of hundreds of thousands of CUIs in Dewesoft by setting their values one line and one CUI at a time. With OrionSIG, creating these .ssi scripts is a fully automated process and engineers save many hours of valuable time. OrionSIG supports testing data for telemetry, or the transmission of data from aircraft and spacecraft to the ground. [7]

---

* Most characters have been replaced with "X" or "0" because actual CUIs cannot be released to the public.
† See *Section IV. OrionSIG Data Types* for a table of all Orion data types like this one.
‡ Most readers will never have heard of SSL because it is a proprietary scripting language by Honeywell Aerospace.
§ For those unfamiliar with decoding binary, "01110001" may translate to the ASCII character "a," the unsigned decimal "97," or many other things depending on what data type it actually represents.
** This superb video features Dewesoft at NASA: http://www.youtube.com/watch?v=22hfUd7NCSE.

II. Software Design

In early June 2013, we started our project with a smaller version of the software requirements in Fig. 5. We expanded them into the current Fig. 5 diagram as we discussed user preferences and needs such as graphing with KSC engineers. Recall that OrionSIG requires an input file and a CUI database to function: it accepts CUIs from the input file, looks them up in the database, and then generates a script based on the results.

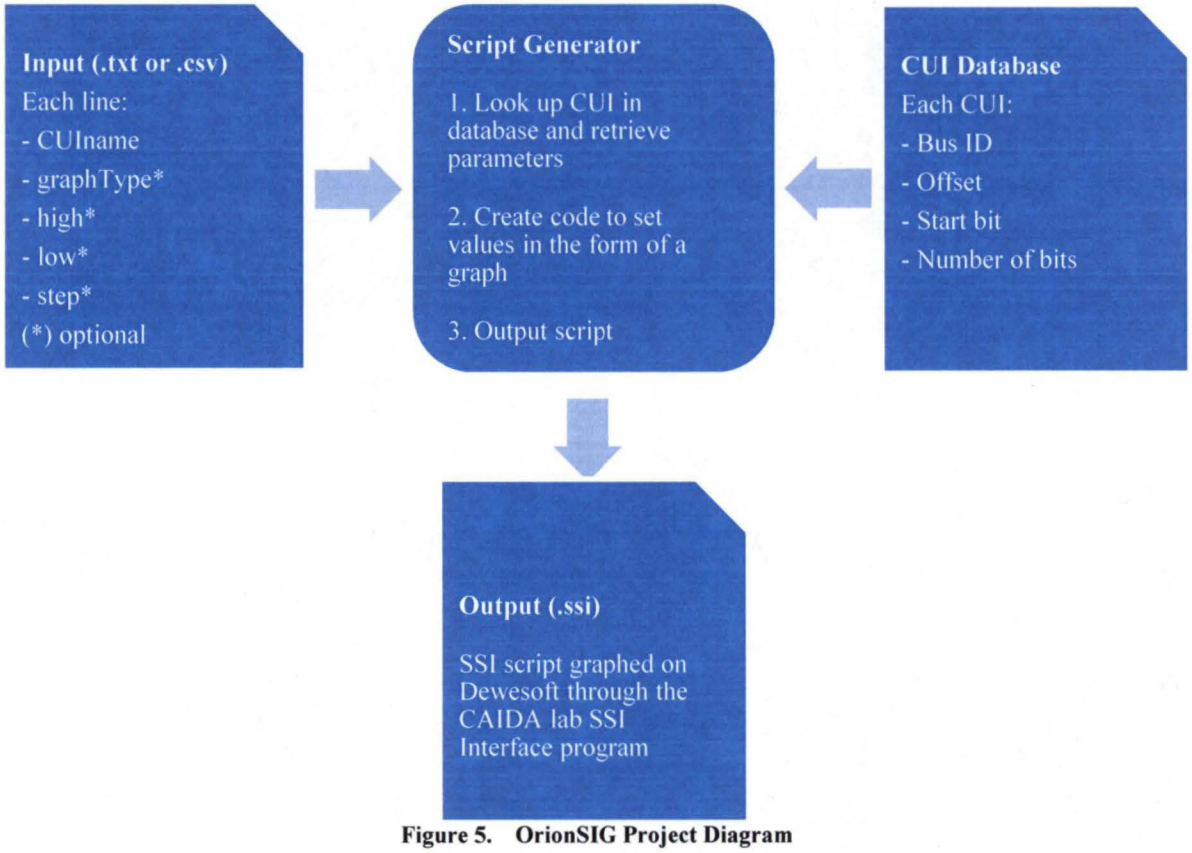


Figure 5. OrionSIG Project Diagram

A. User Input

We placed additional input options in descending order of importance, and graphType belongs at the first position after the CUI name. If OrionSIG is only given a CUI without a graphType or any of the other options specified, it will graph the CUI's values from minimum possible to maximum possible repeatedly in a sawtooth waveform. Many other graph options, however, are common in telemetry and the user can specify another waveform by entering its name one space or one cell to the left of the CUI name.*

The other input options high, low, and step affect which values in the CUI's range from minimum possible to maximum possible are plotted on a Dewesoft graph. Instead of using these default maximum and minimum values, the user can specify different upper and lower limits. Finally, the step value affects how much the graph increments or decrements each second. Step values are constant in most graphs, including sawtooth wave graphs and triangular wave graphs. Curved graphs such as sine and cosine, however, require a step that changes with each increment or decrement and OrionSIG modifies the user-specified step in these cases.

B. Implementation Decisions

We chose to create OrionSIG with iterative software engineering, the C++ programming language, and SQLite database management. OrionSIG was unlike any other project any of us had experienced before because of its massive domain. We initially had no clue of the scope of data we needed to handle or how our software would coexist with Dewesoft, SSL Interface, and other parts of the testing environment described earlier. We started our

* See *Section V. OrionSIG Graph Types* for a table of all OrionSIG graph types.

project with a helpful diagram similar to the one in Fig. 5, but needed to research the answers to many questions, especially regarding the nature of Orion data. Iterative software development fit our extended learning process very well: we created basic software that satisfied our basic understanding of the requirements, learned more about data and user scenarios, integrated this knowledge into the next build, and then repeated the cycle.

The C++ programming language offers a higher level of abstraction that was useful while designing OrionSIG to operate with many different data types. For example, operator overloads*, which do not exist in Java, were instrumental in applying functions to graph different values for CUIs encoded in binary. We also chose C++ because it was our strongest common language. Finally, the SQLite C++ interface is quite well-documented and we were able to incorporate the CUI database into our project easily.

We also considered developing a Graphical User Interface (GUI) version of OrionSIG, but other summer projects took precedence. The benefit of building an OrionSIG GUI also did not seem to justify the cost because our users over the summer simply wanted to submit a spreadsheet of CUIs and then receive script code to graph them for testing. They did not seem as interested in changing the graph parameters from the default: an ascending sawtooth wave.

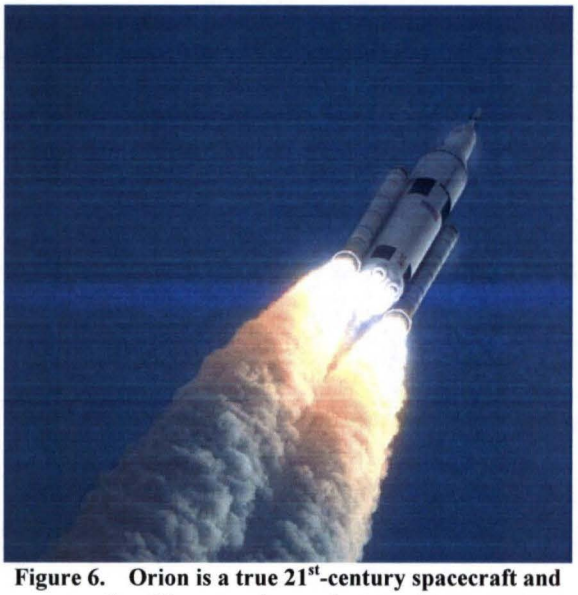


Figure 6. Orion is a true 21st-century spacecraft and supporting this expansive project taught us a great amount about data types (*image courtesy of NASA*).

III. Software Evolution

Finally, we began implementation separately: one of us created a rough C++ program for handling input, calculations, and output while the other created a rough C++ and SQLite program for looking up CUIs in Orion's databases and retrieving needed values. We successfully combined these products early and then spent the majority of the project developing functions to convert between data types and binary as well as vice versa. Recall that we created OrionSIG in iterations: each version accepted more data and flexibility.

Perhaps the most interesting aspect of OrionSIG development involved the SSL output. We were writing code meant to create new code in a scripting language. Working with this lower-level scripting language required some creativity whenever we could accomplish something in a high-level language like C++ but not SSL. Honeywell Aerospace's documentation for this proprietary language was only available to us as a single paper copy in a binder at the CAIDA lab. Some functions were new to us but thorough testing helped us fine-tune our functions and syntax.

We also reduced the running time of our program from over 30 seconds to less than 2 seconds for a sample input file of approximately 100 CUIs (mostly located early in the database) when we discovered the "LIMIT 1" statement in SQLite. This command vastly improved the speed of data retrieval from the CUI database: instead of searching the entire database for multiple instances of a certain CUI, SQLite terminated its search once it had discovered our CUI.

	Initial version	Developed version	Current version
Acceptable data types	Unsigned integers from 0 to 4294967295	Unsigned integers from 0 to 18446744073709551615 Signed integers from -2147483648 to 2147483647	All Orion EFT-1 data types, including chars, doubles, and arrays (See *Section IV. OrionSIG Data Types*)
Input format	A single line from a .txt file	Multiple lines from a .txt file	Multiple lines from a .txt or .csv file

* For those unfamiliar with operator overloading: programmers add new functions that allow them to apply addition, division, and other operators to non-traditional data types, such as strings of binary.

Acceptable graph types	Ascending Descending Rough cosine wave Rough sine wave	Ascending Descending Rough cosine wave Rough sine wave Linear	Many possible options, including refined sine, square, and sawtooth (See *Section V. OrionSIG Graph Types*)
Display behavior	Graphing shows a single period of the wave.	Graphing shows a single period of each wave, advancing through each CUI until the end of the program.	Graphing produces the wave continuously until the user presses any key to advance to the next CUI. The user may also stop the execution between waves.

IV. OrionSIG Data Types

The main challenge of developing OrionSIG lay in creating flexible software that accepts and transfers many types of data, from Boolean values (true or false) to arrays of unsigned long long values (each element is any number from 0 to 18446744073709551615). We also needed to allow bit manipulation for each variable, requiring us to program functions that could convert any of the multiple types of data into binary code. Throughout the project, we explored different methods to optimize the speed of working with the CUI database and long binary numbers. For example, the program handled large binary numbers more efficiently when we stored them as vectors (collections) of Boolean values instead of as vectors of characters or integers.

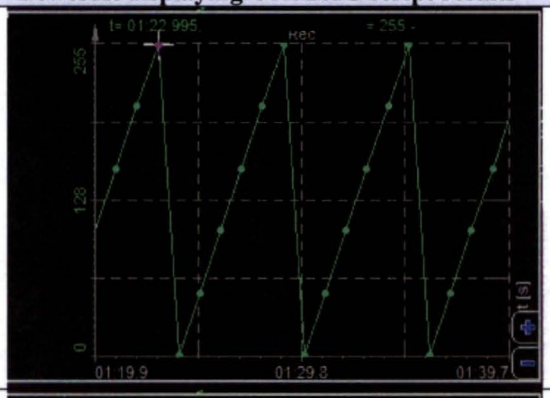
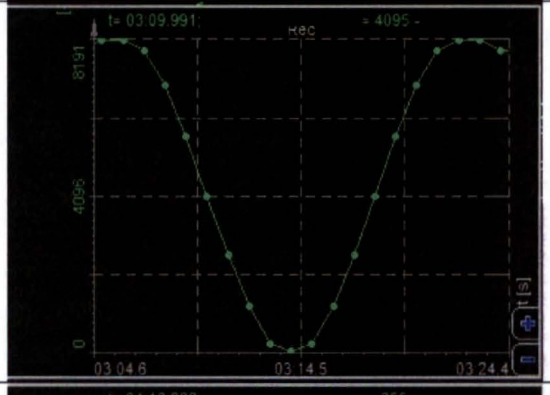
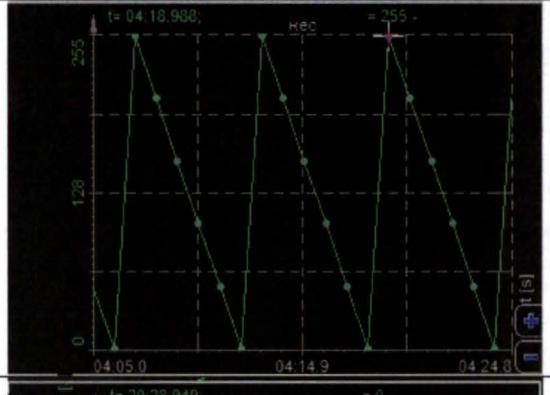
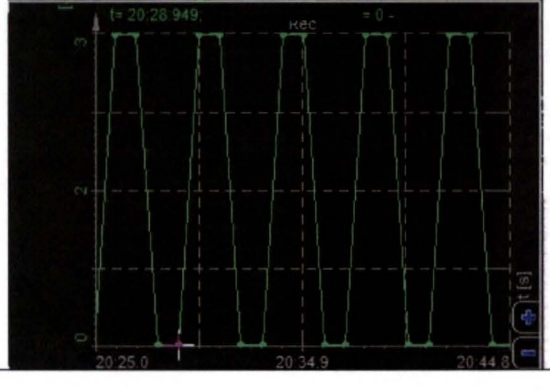
Figure 7. Many data values must be tested to prepare for Orion EFT-1's long journey from launch to splashdown (*image courtesy of NASA*).

Data type	Size in bits	Range of possible values[8]
BOOL	8	false to true
CHAR	8	null character to "Delete" character (all ASCII characters)
SI8	8	-128 to 127
UI8	8	0 to 255
SI16	16	-32768 to 32767
UI16	16	0 to 65535
SI32	32	-2147483648 to 2147483647
UI32	32	0 to 4294967295
UI64	64	0 to 18446744073709551615
FLOAT	64	1.175494351 E – 38 to 3.402823466 E + 38
DOUBLE	128	2.2250738585072014 E – 308 to 1.7976931348623158 E + 308
ENUM	Varies	Varies
ARRAY	Varies	Varies

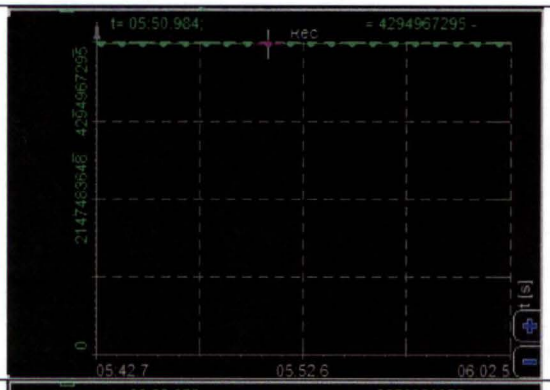
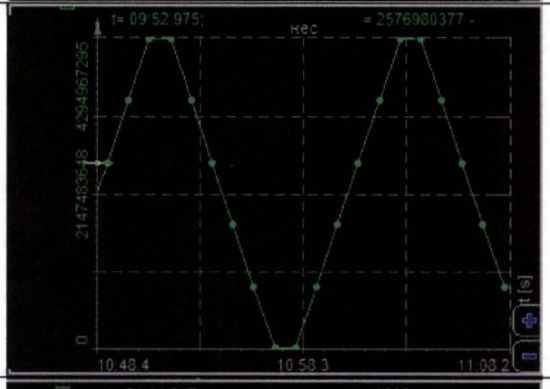
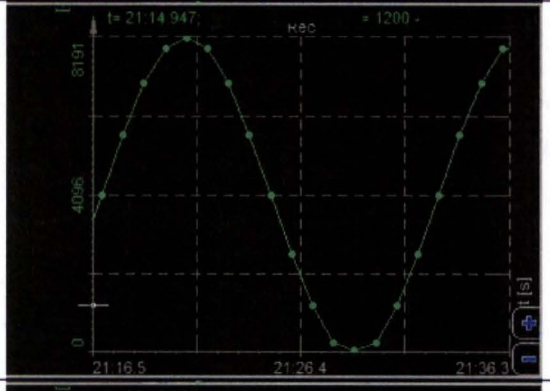
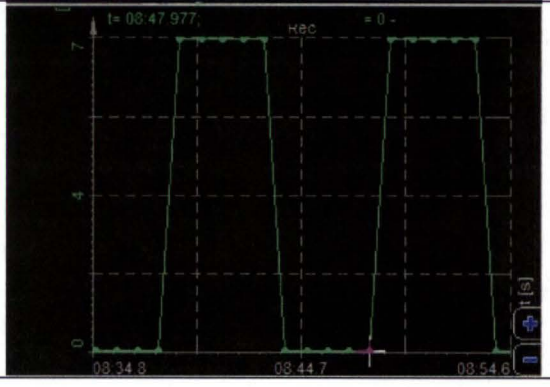
V. OrionSIG Graph Types

OrionSIG's default behavior is to continuously display a current CUI's minimum value and then maximum value, possibly with intermediate values in between depending on the data type. When plotted, these values form an ascending sawtooth wave. Fortunately, users can also add other input on the same lines as each CUI name to request different display options (such as cosine or reverse sawtooth), high values, low values, and interval sizes for generating intermediate values. Developing input validation for all of the possible options in relation with each other was also time-consuming, but OrionSIG's flexibility and helpful handling of user errors was worth the work.

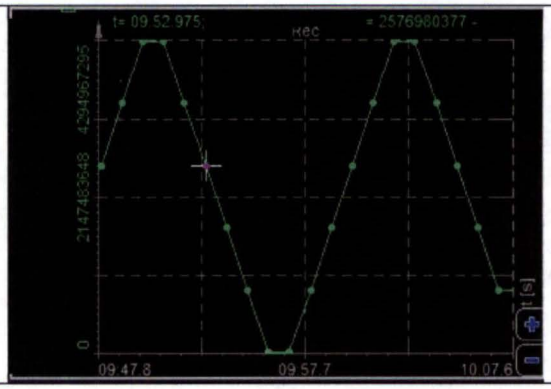
EFT-1 Flight Software currently cannot update more than once per second, so many lines that should be vertical in an ideal graph are slanted instead.

Graph type	Dewesoft displaying OrionSIG script results[*9]
Ascending sawtooth wave	
Cosine wave	
Descending sawtooth wave	
Flip bits	

[*] These images have been edited to protect export-controlled information.

Horizontal line		
Middle triangular wave*		
Sine wave		
Square wave		

* Middle triangle waves are identical to triangle waves, except they originate at the midpoint instead of the low value.

Triangular wave	

VI. Usage

A. Execution Instructions

1. Locate and run OrionSIG.exe.
2. Enter the name of your desired input file (.txt or .csv)*
   a. If your input file is in a different folder, you must enter its full path
   b. Ensure that your input matches the rules in *B. Input Format*
   c. Ensure that the database ("CUI_INFO.db," always included in OrionSIG packages) is in the same folder
3. A SSI script will be generated with the name "[filename]_output.ssi".

B. Input Format

The input file for OrionSIG may contain multiple lines of input, each of which should be formatted as follows:
[CUIname] [graphType] [high] [low] [step]

Parameter	Definition	Example(s)
[CUIname]	Name of the CUI that you wish to evaluate.	OCAVXXX0XXX000XX
[graphType] (optional)	How you want the output script to graph the range of CUI values.	Anything starting with (not case-sensitive): "t" for triangular wave "sq" for square wave "si" for sine wave "m" for middle triangular wave "h" or "l" for horizontal line "f" for flip bits "c" for cosine wave "b", "d", "n" or "r" for backwards / descending / negative / reverse sawtooth Anything else defaults to ascending / forward / increasing / sawtooth / up
[high] (optional)	Highest value to which the output script will set the CUI.	Any value in the range for the data type (such as 0 to 4294967295 for 32-bit unsigned integers)
[low] (optional)	Lowest value to which the output script will set the CUI; can be the same as high in horizontal line	Any value in the range for the data type (such as -128 to 127 for 8-bit signed integers)

* If no filename extension is specified, OrionSIG will first attempt to open a [filename].txt file and then a [filename].csv file.

	graphs.	
[step] (optional)	Rate at which the SSI script will increase / decrease the CUI value.	3 for a high of 5 and a low of -10, resulting in 5 steps per period 107.9 for a high of 3653.8 and a low of 2143.2, resulting in 14 steps per period

VII. Conclusion and Future Development

Orion Scripted Interface Generator (OrionSIG) removes the need for engineers at NASA's Kennedy Space Center to manually write SuperScript Language code for testing and graphing Orion spacecraft telemetry measurements. Fortunately, OrionSIG can also help test data for any other mission that uses CUI databases in the same format. A GUI version of OrionSIG may be able to include an option to generate databases of CUIs from existing Excel spreadsheets to quickly enable this testing model for CUIS from any mission. This GUI version could also allow users to apply options to ranges of CUIs inside the input. For example, a user could change the graphType for CUIs 11 through 21 to "triangular wave". We hope our comprehensive data test tool holds great potential for supporting future NASA efforts and wish the Orion team all the very best for a successful EFT-1 mission.

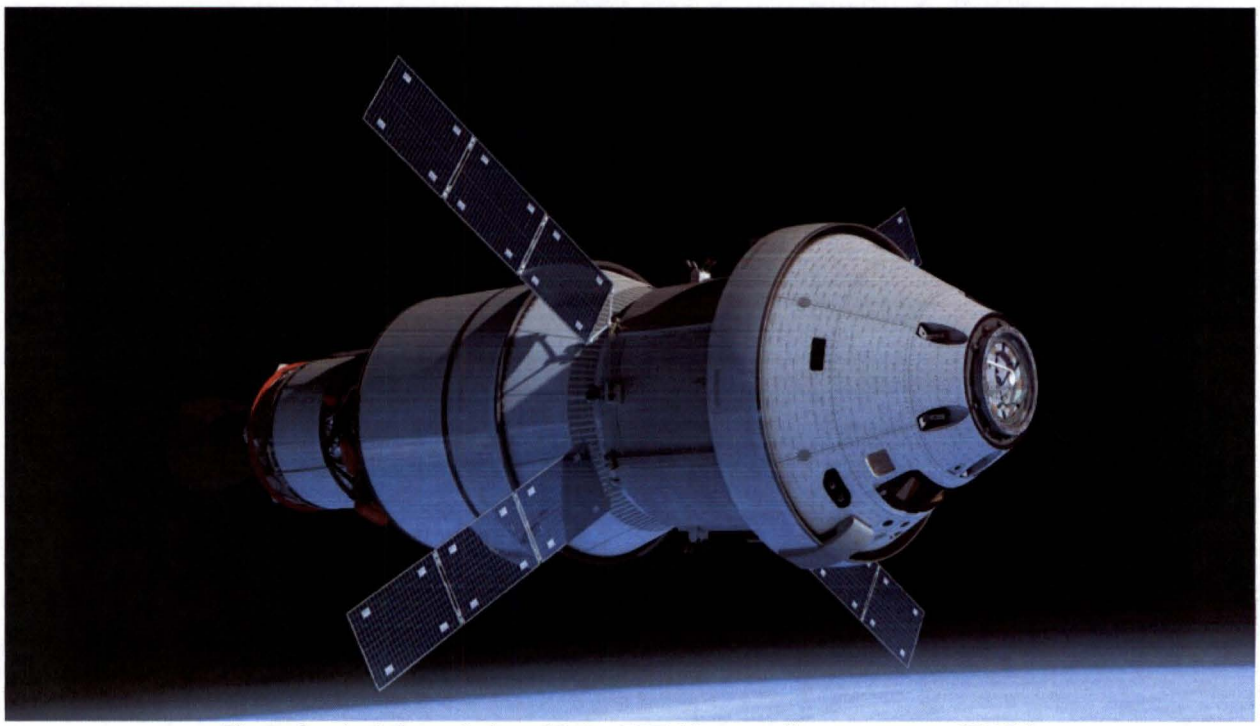


Figure 8. Artistic rendering of Orion EFT-1 in space (*image courtesy of NASA*)

Acknowledgments

My summer 2013 NASA internship exposed me to large-scale software engineering, profound thoughts about the universe as well as human space exploration, and many wonderful people who I will remember forever. For this fantastic experience, I am extremely grateful to first and foremost, ACCESS (Achieving Competence in Computing, Engineering, and Space Science), which funded my internship program. I also thank my university, Rochester Institute of Technology and especially their Computer Science department, for bringing me from knowing nothing about programming a few years ago to where I am today. Several NASA employees deserve a very special thank you for helping me communicate, understand my work, and succeed even though I am profoundly deaf and American Sign Language is my primary language. Thank you to my mentor Curtis Williams; my fellow Avionics interns Samuel Harris, Uriae Walker, and Anthony Castrovillo; and KSC sign language interpreters Stephanie

Watkins, Laurie Carter, Donna Fisher, Kimba Conner, and Jennifer Rogers. OrionSIG and my knowledge would not be where they are today without these individuals.

I am also very grateful to Arnold Postell for leading a brilliant NASA Avionics Division with the most friendly workforce I have ever met; Brian Luther for running an excellent Guidance, Navigation & Flight Controls Branch and providing me with the opportunity of a lifetime; Dean Orr for bringing the interns the best NASA has to offer; Glenn Rosenthal for teaching me telemetry; Rose Austin, Benita DeSuza, and Grace Johnson for coordinating a terrific KSC Education Office as well as many intern events; Stephanie Stilson, Mike Ciannilli, Patrick Feeney, and Marcelo Dasilva for cultivating a KSC LGBT Employee Resource Group with exciting potential; Caren Ensey for bringing me up to speed on NASA software engineering standards; and Annette Pitt for answering my many questions.

Last but not least, I also want to sincerely thank those who made me feel welcome at NASA and Cape Canaveral in at least some small way, whether through words or actions. I was only here for ten weeks and wish I could spend more time with these people, but they made a big difference when I moved to a completely new area of the country without knowing anyone there: Felix Pena, Sherry O'Bryan, Kaley McCarty, Kenneth Jenkins, Charles Parrish, Alex Tsoras, Karl Stolleis, Hali Jakeman, Ashley Williams, Joanna Johnson, Steven Pancoast, Kathy Meesakul, Christine Okrepkie, Edward Tugg, Robert Zambrana, Nicole Delvesco, Jessica Conner, Kristie Durham, and many others.

References

[1]NASA. "NASA's Orion Moves Closer to Next Giant Leap." *www.nasa.gov* N.p., 8 March 2012. Web. 18 July 2013.
[2]NASA. "Orion Quick Facts." *www.nasa.gov* N.p.. Web. 19 July 2013.
[3]NASA. "Apollo 17." *www.nasa.gov* N.p.. Web. 19 July 2013.
[4]NASA. "Six Flags on the Moon: What is Their Current Condition?" *www.nasa.gov* N.p., 21 April 2012. Web. 19 July 2013.
[5]NASA. "NASA On Course to Launch Orion Flight Test." *www.nasa.gov* N.p., 28 February 2013. Web. 18 July 2013.
[6]Dewesoft. "About us." *www.dewesoft.com* N.p.. Web. 23 July 2013.
[7]Rosenthal, Glenn. "A Course in Telemetry." *NASA KSC Rocket University*. Cape Canaveral, FL. July 9-11, 2013. Lecture.
[8]Microsoft. "Storage of Basic Types." *Microsoft Developer Network* N.p. Web. 24 July 2013.
[9]Dewesoft, Software Package, Ver. 7.0.5. Dewesoft, Trbovlje, Slovenia. 2012.