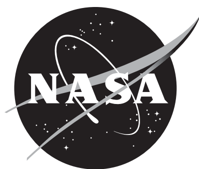
View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by NASA Technical Reports Server

NASA/TM—2014-218319



Advanced Extravehicular Mobility Unit Informatics Software Design

Theodore W. Wright
Glenn Research Center, Cleveland, Ohio

June 2014

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

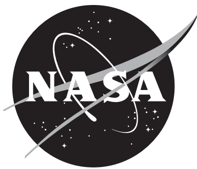
- E-mail your question to *help@sti.nasa.gov*

- Fax your question to the NASA STI Information Desk at 443–757–5803

- Phone the NASA STI Information Desk at 443–757–5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM—2014-218319



Advanced Extravehicular Mobility Unit Informatics Software Design

Theodore W. Wright
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2014

Trade names and trademarks are used in this report for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Available electronically at http://www.sti.nasa.gov

Advanced Extravehicular Mobility Unit Informatics Software Design

Theodore W. Wright
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

This is a description of the software design for the 2013 edition of the Advanced Extravehicular Mobility Unit (AEMU) Informatics computer subassembly. The Informatics system is an optional part of the AEMU space suit assembly. It adds a graphical interface for displaying suit status, timelines, procedures, and caution and warning information. In the future it will display maps with GPS position data, and video and still images captured by the astronaut.

Contents

1 Introduction 1

2 Hardware Platform 2
- 2.1 Real Time Clock . . . 4
- 2.2 Compass and Accelerometer . . . 4
- 2.3 GPS . . . 4

3 Operating System and Software Libraries 5
- 3.1 Linux . . . 5
- 3.2 Qt . . . 6
- 3.3 ZeroMQ . . . 6
- 3.4 Google Protocol Buffers/Nanopb . . . 7

4 Software Architecture 8
- 4.1 C++ Infrastructure . . . 8
  - 4.1.1 Setup . . . 9
  - 4.1.2 Periodic Processing . . . 10
  - 4.1.3 Sending Data . . . 10
  - 4.1.4 Receiving Data . . . 11
  - 4.1.5 AppData Data Exchange . . . 11
- 4.2 QML/Javascript Graphical User Interface . . . 12
  - 4.2.1 Consumable Display . . . 13
  - 4.2.2 Communications Display . . . 13
  - 4.2.3 Buddy Display . . . 15
  - 4.2.4 File Selection . . . 16
  - 4.2.5 Procedure Display . . . 17
  - 4.2.6 Timeline Display . . . 18
  - 4.2.7 Caution and Warning Display . . . 19

5 Summary 20

6 References 20

1 Introduction

This is a description of the software design for the 2013 edition of the Advanced Extravehicular Mobility Unit (AEMU) Informatics computer subassembly. The Informatics system is an optional part of the space suit assembly. It adds a graphical interface for displaying suit status, timelines, procedures, and caution and warning information. In the future it will display maps with GPS position data, and video and still images captured by the astronaut.

Note that the Informatics system is a work in progress, and many of the features, interfaces, telemetry formats, timelines, procedures and other data are currently just placeholders to test that the design is working as expected. This document has been updated to describe the newer Informatics system as used in the subassembly integration tests.

2 Hardware Platform

There are currently no space-rated radiation-hardened computer systems with a Graphics Processing Unit (GPU). Until an affordable graphics capable computer with a path-to-flight is available, for breadboard testing Informatics will use a simple, low cost, low power, embedded computer that is hopefully representative of future space-rated hardware.

The hardware selected is called a *Beaglebone Black*. It is a small (3.4" by 2.1"), light (1.4 oz.), $45 computer board with an ARM 7 architecture processor and PowerVR graphics coprocessor combined in a System-On-Chip package. It has 512 MB of RAM memory, a 2 GB flash file system and a SD card slot for the operating system and user file systems, 100 Mbit/s Ethernet, and a USB host connector. The processor has a maximum speed of 1 GHz, but if the software load permits, it slows to 300 MHz to reduce power consumption.

The Informatics system receives inputs over an Ethernet network. This could eventually include audio information (for voice recognition) and video information (for real time video displays), but those are not implemented in the current Informatics software. Reception of streaming audio over Ethernet has been successfully tested with the Informatics software, but the audio was just stored to a file (voice recognition is not implemented).

With the addition of an HDMI to LVDS converter board, the Beaglebone can drive the cuff display (shown in Figure 1) developed for the NASA Desert Research and Technology Studies (*Desert RATS*) tests. The 1024 by 600 resolution 5" cuff display has edge buttons to use for operator control, and the buttons connect through a USB keyboard interface. For bench top testing, the Beaglebone can drive an *LCD7* 800 by 480 resolution 7" LCD display that mounts on top of the Beaglebone. The LCD7 has five pushbuttons that emulate keystrokes for operator input (and a resistive touchscreen, but that will not work if the operator is wearing gloves).

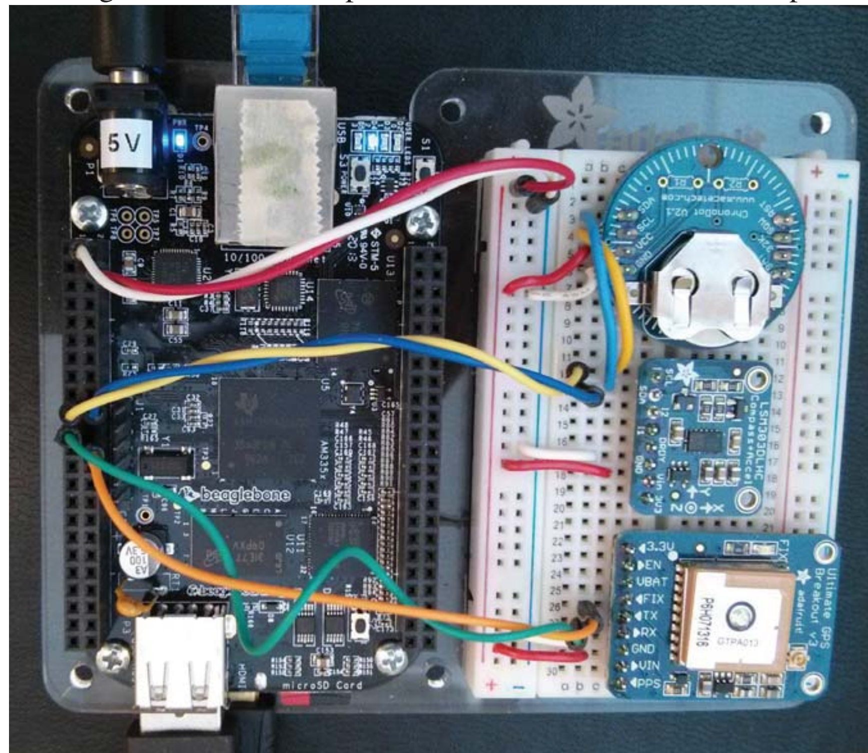
The Beaglebone has expansion connectors that can add optional additional hardware. Hardware and software support has been tested for the following expansions:

- Real Time Clock
- Compass and Accelerometer
- GPS

A Beaglebone Black computer with breadboard connections for the hardware expansions is shown in Figure 2.

Figure 1: Beaglebone Black computer driving the Desert RATS cuff display

Figure 2: Beaglebone Black computer with breadboard Clock, Compass and GPS



2.1 Real Time Clock

The Beaglebone hardware does not include a battery-backed real-time clock circuit, so when the system is powered on, the operating system time defaults to January 1, 1970. The correct time and date can be set manually or with a Network Time Protocol daemon if the appropriate server is available (the Radio subassembly provides this, if present). Adding a battery backed clock chip enables correctly time stamped data under all configurations without user intervention.

A battery-backed clock based on the Maxim DS3231 temperature compensated chip is easily connected with a 2 wire (plus 3.3V power and ground) I2C interface. An I2C interface is preconfigured on a stock Beaglebone Black on P9 expansion header pins 19 and 20. The clock appears at address 0x68. A Linux driver for the compatible DS1307 chip comes with Angstrom Linux distribution, so the clock can be configured and read by adding two lines to a system startup script as shown in Listing 1.

Listing 1: Configuring the clock driver and reading the clock

```
# create /dev/rt1
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device

# read battery backed hardware clock
hwclock -r -f /dev/rtc1
```

2.2 Compass and Accelerometer

The Informatics display has a compass heading indicator to help astronauts orient themselves for surface operations. For testing on Earth, this can be driven by an electronic compass. The breadboard design uses an STMicroelectronics LSM303 chip containing a 3 axis magnetometer and a 3 axis accelerometer with an I2C interface (with addresses 0x1e and 0x19 respectively). Vector cross products can be used to combine the magnetic field vector with the gravity vector to determine a compass heading without requiring a fixed orientation of the subassembly.

There is no pre-written device driver for this chip, so a Python language script is started in the background that reads the I2C registers to determine magnetic field and gravity vectors, combines them to generate a compass heading, and publishes the heading using the ZeroMQ/Protocol Buffer format described in Sections 3.3 and 3.4.

2.3 GPS

The Informatics display also has an astronaut position display, which appears as latitude, longitude and altitude for Earth testing. This information comes from GPS satellites, using a GlobalTop MTK3339 GPS chipset connected through a 2 wire (plus power and ground) serial interface.

The Beaglebone Black P9 header pins 21 and 22 can be configured as a serial UART using the Linux device tree overlay support. Pre-compiled device tree overlays for the Beaglebone UARTs come with Angstrom Linux, so all that is needed to make the UART device appear is to write the device name to the configuration register at system startup, as shown in Listing 2.

Listing 2: Configuring the Serial UART connected to the GPS

```
# create serial port /dev/tty02
echo  BB-UART2 > /sys/devices/bone_capemgr.8/slots
```

There is no pre-written device driver for a serial GPS, so a Python language script is started in the background that opens the serial port and parses the NEMA formatted text data stream to extract latitude, longitude and altitude. It then publishes the data using the ZeroMQ/Protocol Buffer format described in Sections 3.3 and 3.4.

3 Operating System and Software Libraries

Although the software environment for Informatics is geared for embedded system compilers and tools, it also tries to take advantage of existing higher level open source protocols, libraries, and operating systems. This allows the software to be developed in a more timely manner, and with rapid prototyping several ideas can be tried and discarded to determine the better way to do things.

3.1 Linux

There are no hard real-time software requirements for Informatics, so Linux was chosen as the operating system due to its ease of use, good development tools, documentation, hardware support, and price (free).

The Beaglebone Black was delivered with the Angstrom Linux distribution on it's flash file system. This was updated to the latest version (Angstrom release August 21, 2013 - Linux kernel version 3.8.13) before testing, and run from a 16 GB high speed micro SD card to provide ample free file system space. Ubuntu Linux 13.04 was also tested, and was a little bit easier to work with due to primarily to past experience and familiarity. However, it was decided to use Angstrom for the present because it has drivers that support the hardware accelerated GPU, and Ubuntu does not. Drivers could probably be compiled for Ubuntu with some effort, but the time has not been available to complete that task.

Several standard Linux software daemons are important for Informatics. The Beaglebone has no battery backed clock, so it relies on a Network Time Protocol daemon to set its clock and keep itself synchronized with other assemblies. The radio subassembly will provide an NTP server (Informatics could also use a GPS chip for this purpose, or a dedicated battery-backed clock chip). A Secure Shell daemon allows logging into the Beaglebone across the network, and it allows file

transfer to and from the Beaglebone with the Secure Copy Protocol. A File Transfer Protocol daemon could also be used to update files on the Beaglebone's SD card. The GStreamer software is used to receive and decode audio streams over Ethernet from the Audio subassembly.

3.2 Qt

The previous generation Informatics system ran on an Intel computer and used the Adobe ActionScript (Flash) graphics development tools and libraries. Flash is being discontinued and does not support non-Intel platforms, so a different Graphical User Interface (GUI) library is required.

The library chosen is the open source *Qt* library. Qt was primarily a C++ based tool, but recent versions have added a declarative layout language with embedded Javascript called *QML*. QML is similar to ActionScript, and it is much faster to work with than C++. GUI development is tedious and time consuming, so faster development speed is helpful. Qt has several other advantages:

- It is cross-platform. This allows code to be developed on desktop Linux, Macintosh, or Windows computers and simply recompiled to run on the embedded computer, where it looks and runs identically.
- On embedded computers, Qt interfaces directly with the video driver, bypassing operating system layers such as the X11 Window system. It has good performance even on relatively slow hardware.
- QML layouts can adapt somewhat to different screen resolutions, allowing easy hardware changes.

Qt version 4.8 was installed from the Angstrom software repository and used for all Informatics development so far. However, Qt version 5 was recently released and claims to be even more optimized for embedded systems. The Ubuntu 13.04 Linux release that was tested used Qt 5.0, but did not include the necessary accelerated hardware support. A switch to Qt 5 should be re-evaluated when support matures.

3.3 ZeroMQ

A *Publish/Subscribe* network architecture is a best practice for reliable information exchange between loosely coupled networked subassemblies, such as the AEMU. While it is possible to create a simple system based on UDP broadcasts, past experience has shown that making it reliable and fixing all the corner cases can become complicated.

The open source *ZeroMQ* socket library makes coding a basic Publish/Subscribe system trivially easy, requiring about a dozen lines of code. ZeroMQ is a library that is linked to code, so unlike most other "middleware" solutions, it does not require a separate daemon process or broker software. It

was originally developed for high frequency trading applications on Wall Street, where efficiency is paramount.

There are dozens of language bindings to ZeroMQ. The Python binding is used to create a simple "fake telemetry" generator that is used to make network traffic to test the Informatics software. Another small Python/ZeroMQ script can be used to decode and print network traffic generated by AEMU subassemblies. ZeroMQ version 3.2.2 is currently used by Informatics.

3.4 Google Protocol Buffers/Nanopb

ZeroMQ is responsible for moving bytes between computers, but does not care what format those bytes take. A *wire protocol* is needed to standardize the format of the data being exchanged. The open source *Google Protocol Buffer* format was selected as a binary wire protocol.

Advantages of using Protocol Buffers include:

- An efficient (slightly compressed) binary data representation
- Easy backwards compatible extension as more data is added
- Thoroughly tested and optimized by Google
- A code generator that creates the necessary encoding and decoding software from a simple data description language
- Portable generated source code is added to projects instead of requiring a library

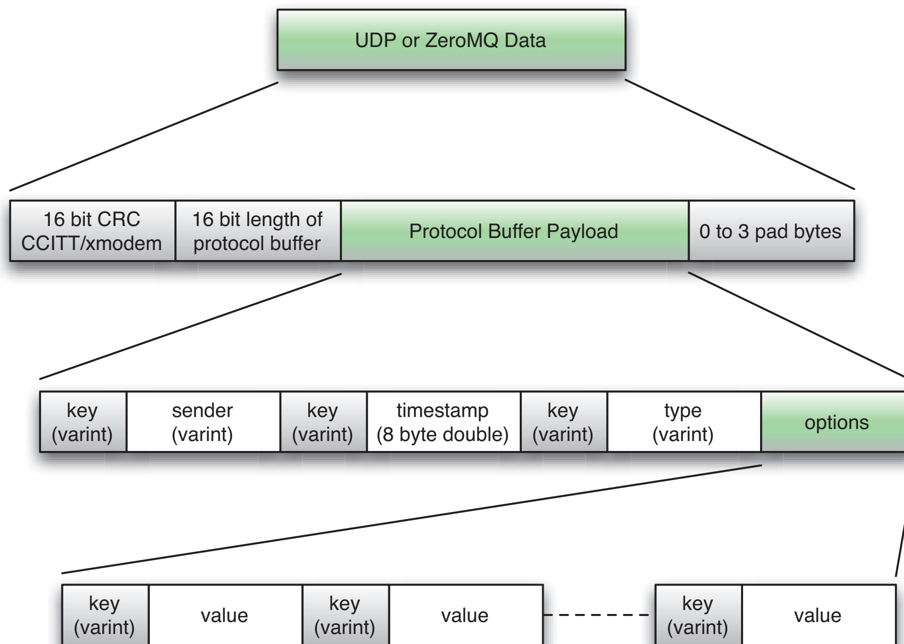
Protocol Buffers as packaged from Google has support for the C++ and Python languages, but implementations for other languages are provided by third parties.

Informatics and the other AEMU subassemblies use the Nanopb C language implementation. Nanopb is designed for embedded systems, and an option to statically allocate all of its memory. Although the Informatics code is partially written in C++ and could use the higher level interface, it uses the Nanopb implementation to share commonality with other AEMU assemblies.

The wire format of the data published by ZeroMQ is shown in Figure 3. A 16 bit Cyclic Redundancy Check is stored in the first part of the message (and is computed over the remainder of the message). Next is two bytes that encode the length of the Protocol Buffer part of the message. The CRC and length are sent in network byte order. Next is the Protocol Buffer payload, followed by 0 to 3 pad bytes (ASCII 0x00) to make the total message length a multiple of 4 for easier compatibility with the older ARM architecture CPU used in the Audio subassembly.

The Protocol Buffer port of the message has 3 mandatory fields followed by 0 or more optional fields. The mandatory fields are *sender*, *timestamp*, and *type*. The *type* field determines which optional fields follow. For example, the *EVENT_LOCATION* type will always be followed by the *latitude*, *longitude* and *altitude* fields. About 2 dozen message types are currently understood by the Informatics software.

Figure 3: ZeroMQ/Protocol Buffer wire format



4 Software Architecture

The software architecture of the Informatics GUI is split into two parts:

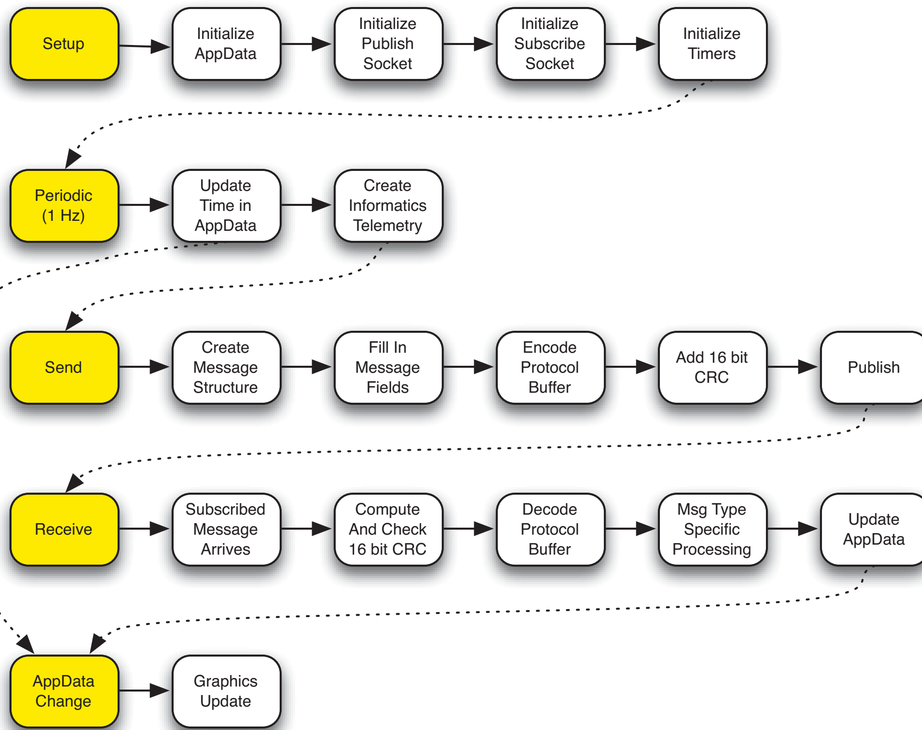
- An *infrastructure* part written in the C++ programming language that is responsible for all of the network communication and network packet decoding and verification.
- A graphical user interface part written in the Qt QML language (which is a combination of a declarative graphics layout language and Javascript) that is responsible only for formatting the display of data.

The architecture uses a software component called *AppData* that handles the exchange of data between the C++ and QML halves. AppData hides the details of the Qt Signal and Slot update mechanism, and makes it so that when new telemetry arrives and the infrastructure code changes a value, updates are propagated only to dependent data in the user interface.

4.1 C++ Infrastructure

A block diagram of the C++ infrastructure code is shown in Figure 4. The infrastructure is divided into five major operations: *Setup*, *Periodic Processing*, *Sending Data*, *Receiving Data*, and *AppData Change*. Major operations are broken into a number of minor operations, and solid arrows between them show their order of execution. Some minor operations will trigger the start of

Figure 4: Block Diagram of C++ code functions



different major operations, and these links are indicated with dotted arrows.

4.1.1 Setup

The *Setup* code is executed when the program starts. Its first job is to initialize AppData, and since AppData is actually a class, it is initialized by creating an instance and then calling the *setter functions* for the class members that need to have initial values.

Next a ZeroMQ socket is created for publishing any information that Informatics needs to make available to other AEMU assemblies.

A different ZeroMQ socket is then subscribed to any data Informatics needs to receive from the network. A single ZeroMQ subscribe socket can be connected to multiple ports and interfaces, so only one subscribe socket is needed. The subscribe socket is connected to the port that the publish socket uses, so anything Informatics publishes will be received and processed in the same manner as data from other AEMU assemblies.

The last step of the Setup code is to initialize timers. After initialization, a 1 Hz timer triggers *Periodic Processing* at regular intervals.

A second unrelated timer exists in the infrastructure code for keyboard handling. If a key is held

for more than 800 ms before it is released, the *Alt* flag is added to the keystroke, allowing it to be treated differently in the user interface code. This is intended to help work around the limited number of input buttons available on some displays.

4.1.2 Periodic Processing

The *Periodic Processing* runs at 1 Hz intervals. The first thing it does is to get the current time and update the value in AppData. This change in AppData will trigger an update to GUI elements that depend on time.

The second periodic task is to create and publish a telemetry packet containing the Informatics status, which so far is always *nominal*. The process for creating and publishing data is the same for any type of telemetry, and is described in the next section.

The final periodic task is to create and publish a telemetry packet containing Buddy Information (suit ID, consumable time remaining, position, and caution or warning messages).

4.1.3 Sending Data

Publishing data starts with allocating a Nanopb protocol buffer structure (a type of "C struct") to hold the details of the packet being created. A call to the standard C *memset* function is used to initialize all fields in the structure to zero.

Next, values are assigned to some of the fields in the structure. There are currently three mandatory fields that must always have values: *sender*, *type*, and *timestamp*. The *sender* field indicates that this data is coming from Informatics. The *type* field contains the type of the protocol buffer message, which determines the additional optional fields that must be included. The *timestamp* field contains the time the packet is created.

Optional data fields may be really be required, depending on the *type* field. For example, if the *type* fields say that the packet contains Informatics Status information, then a value must be assigned to the *status* field.

Because this is the simple Nanopb interface instead of the more full featured C++ interface for protocol buffers, an additional field of the form *has_X* must be set to *true* for every optional parameter *X* to indicate that it is being used.

After the fields are filled in, the structure is encoded as a protocol buffer into an array of bytes. Two function calls are required to perform this task: *pb_ostream_from_buffer* and *pb_encode*.

After encoding, the encoded length is prefixed to the beginning of the encoded data, and 0 to 3 pads bytes are added to the end.

Next a 16-bit Cyclic Redundancy Check (CRC - CCITT xmodem variant) is computed for the array of bytes, and prepended to its beginning.

Finally, the byte array is sent to the ZeroMQ publish socket for delivery to the network. Note that since the Informatics subscribe socket is connected to the port of the publish socket, anything published will trigger the reception of a network packet.

4.1.4 Receiving Data

The *Receiving Data* operation is started whenever a complete packet has arrived on the subscribe network socket. A 16-bit CRC is computed for the packet (ignoring the first two bytes), and then compared to the 16-bit CRC assumed to be stored in the first two bytes of the packet. If they are not identical, the packet is discarded.

After the CRC is stripped off, the next two bytes are read to find the length of the Protocol Buffer portion of the packet, which is then extracted from the bytes immediately following the length. The length is required because there may be pad bytes at the end that must be ignored.

Next, the Protocol Buffer bytes are decoded into a Nanopb protocol buffer structure. If there is a decoding error (because it somehow received a packet not representing a Protocol Buffer), the packet is discarded.

The *type* field of the structure is examined, and a case statement chooses some type specific code to execute based on the field. In almost all cases, this code simply reads the optional fields required by the type from the structure and calls the *setter functions* to write the values to AppData.

Data handling for the Caution and Warning type is slightly different, because that data is maintained in a separate list structure that works much like AppData, but is variable length (there may be zero or more Caution and Warning messages that have been received). Items are never deleted from this list, but their timestamp and acknowledged status is updated with the most recently observed telemetry.

4.1.5 AppData Data Exchange

The *AppData* class is basically just a place to store values (as instance variables) with *getter* and *setter* functions for reading and writing each value. The *setter* function checks to see if the value has changed, and if so, emits a Qt *signal* to notify dependent code that it needs to update. The signal declaration and a Qt *property* declaration are also part of the class.

This results in five lines of hard to read boilerplate declarations and repetitive code for each value that is stored in AppData, when all a programmer really cares about is the variable's type and name. To help make the code maintainable, the repetitive code is generated from an M4 macro expansion. Listing 3 shows the code generated from a line with the *APPDATA(double, timeNow)* macro that contains only the type *double* and name *timeNow*. This expansion allows the AppData class definition to consist of an easy to read macro line for each variable.

A *prebuild.sh* script is run before compilation to perform M4 macro expansion (resulting in the file called *qmlapplicationviewer.h*) and to generate the C code for protocol buffer encoding and decoding from the protocol buffer definition file (*aemu.proto*).

Listing 3: M4 macro expansion output for input: *APPDATA(double, timeNow)*

```
    // autogenerated from qmlapplicationviewer.in — DO NOT EDIT
Q_PROPERTY(double timeNow READ timeNow WRITE settimeNow NOTIFY timeNowChanged)
public: void settimeNow(const double &s) { if (s != m_timeNow) { m_timeNow = s; emit timeNowChanged(); }}
double timeNow() const { return m_timeNow; }
signals: void timeNowChanged();
private: double m_timeNow;
```

4.2 QML/Javascript Graphical User Interface

The layout of the graphical user interface is based on previous version of the Informatics interface as tested at Desert RATS and user interface prototypes from AEMU human factors studies.

The graphical user interface has copious information to display and a limited screen area on which to display it, so the screen is divided into a number of areas where the largest area varies to show details of a particular data set. There is a row of tab-like buttons across the top of the display that are used to select the variable set of data to display. On the right edge of the display is an area of persistent information that is always displayed, no matter which tab is selected along the top. The bottom portion of the persistent information is a duplicate of the information on the 2 line text display controlled by the CWCS subassembly. See Figure 5 for an example.

The currently selected tab across the top is highlighted with a blue border. There are several ways of switching to a different tab to accommodate running the program on different types of hardware. If a keyboard is connected to the hardware, the left and right arrow keys will select adjacent tabs. The LCD7 display on the Beaglebone development board has 5 pushbuttons (*up, down, right, left*, and *enter*), where the left and right buttons switch between tabs. If a mouse is connect to the hardware, the tab can be clicked upon to select it. If the hardware has a touch-screen interface (like the LCD7) touching the tab on the display will select it.

Some data sets have a secondary menu of operations that appears along the left edge of the display. For example, Figure 8 shows the File Selection display (because the FILE tab is selected at the top), which has four operations in the menu: *Scroll Down, Scroll Up, Select File,* and *Up Level*. The currently selected operation is highlighted with a yellow border, and can be changed with the up and down arrow keys, dedicated up and down buttons, mouse click or touch. The *enter* key or pushbutton is used to perform the action associated with the currently selected menu item. With a mouse or touchscreen, selecting an already highlighted menu item will perform its action.

Each of the data set displays that have (partial) implementations will be described in greater detail. However, the user interface is a work in progress, and some of the displays so far are just simple proof of concept implementations with placeholders for many of the details.

Several Informatics displays are considered less critical for near term implementation, so they are present in the GUI only as placeholders. These include:

- CAM (Camera Display, currently blank)
- MAP (Map Display, currently blank)

4.2.1 Consumable Display

There are four consumable displays: *Primary Oxygen, Secondary Oxygen, Battery Charge* and *Water*. Each of these displays has a tab at the top of the screen to select it, labelled: *O2, S_O2, BATT,* and *H2O*, respectively. Figure 5 shows the Primary Oxygen display. Under the name on the tabs is the estimated time (in hours and minutes) before the consumable is exhausted, with the limiting (nearest) time colored red for emphasis. Secondary Oxygen is not consumed until Primary Oxygen is gone, and its time will show 23:59 until it is being consumed (this is the largest number that can be represented in this implementation).

When any of the four consumable displays are shown, the left side of the display will show bar graphs for all four consumables. Numbers near the graphs show the current level of the consumable, the maximum value, the percent remaining, as well as the time remaining. If the current value is less than some critical value, the bar graph will turn from green to red, and the critical value will be displayed inside the graph. All of these numbers, with the exception of percent remaining, are received via telemetry from the Caution and Warning Control System (CWCS) subassembly. They are not fixed values in the GUI code.

The center of the screen will show consumable specific details (that vary when a different consumable tab is selected), but currently is is just a placeholder.

Under the consumable specific display is biomedical data, currently consisting of heart rate and metabolic rate. Each bar in the metabolic rate chart represents half an hour.

4.2.2 Communications Display

The Communication Display (shown in Figure 5) consists mainly of temporary placeholder areas for the four major subsystems: Informatics, Radio, Audio, and CWCS. These areas turn green if a recent network message arrived indicating that the subsystem status is nominal. The status and last update time of the status for each subsystem is shown as text.

The Informatics area also displays a count of the total number of network messages received, the number of CRC decoding errors, and the number of protocol buffer decoding errors.

At the bottom of the Communication Display is an area showing received text messages (sent from ground stations or other assets). When a new text message arrives, the message list scrolls to the end to show the new message and the COMM tab at the top of the screen turns blue as a visible

Figure 5: Consumable Display

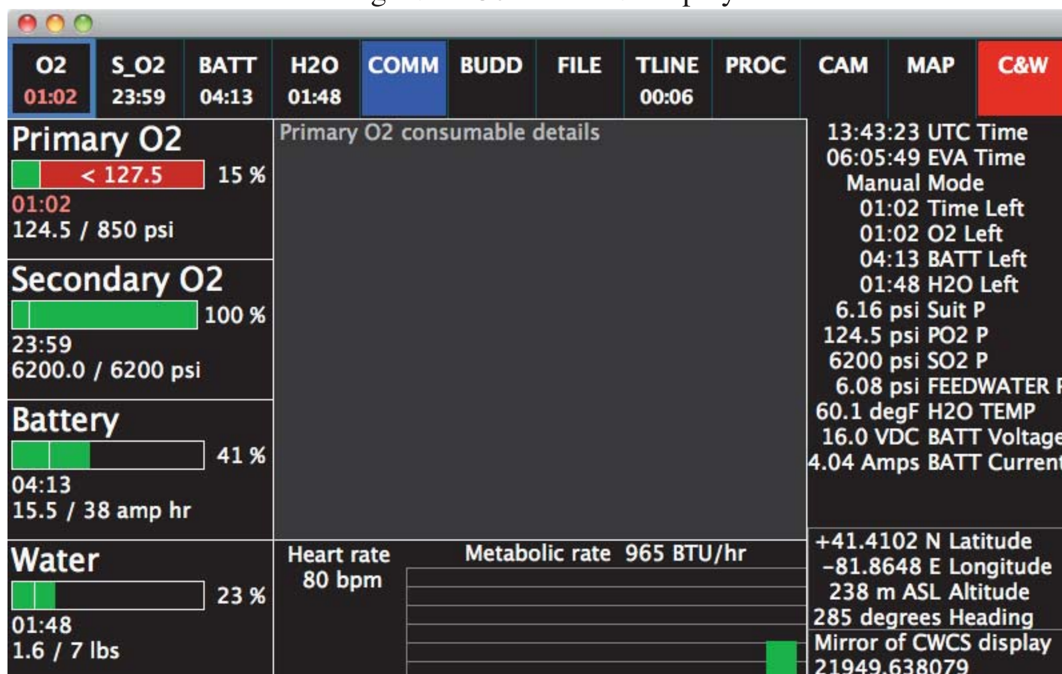


Figure 6: Communications Display

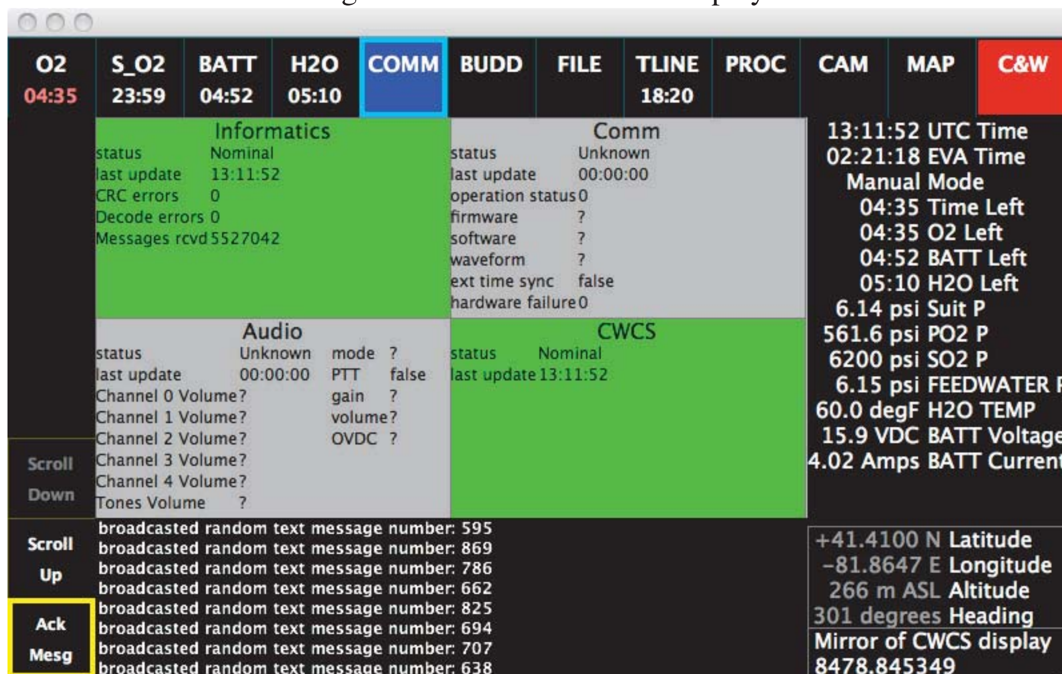
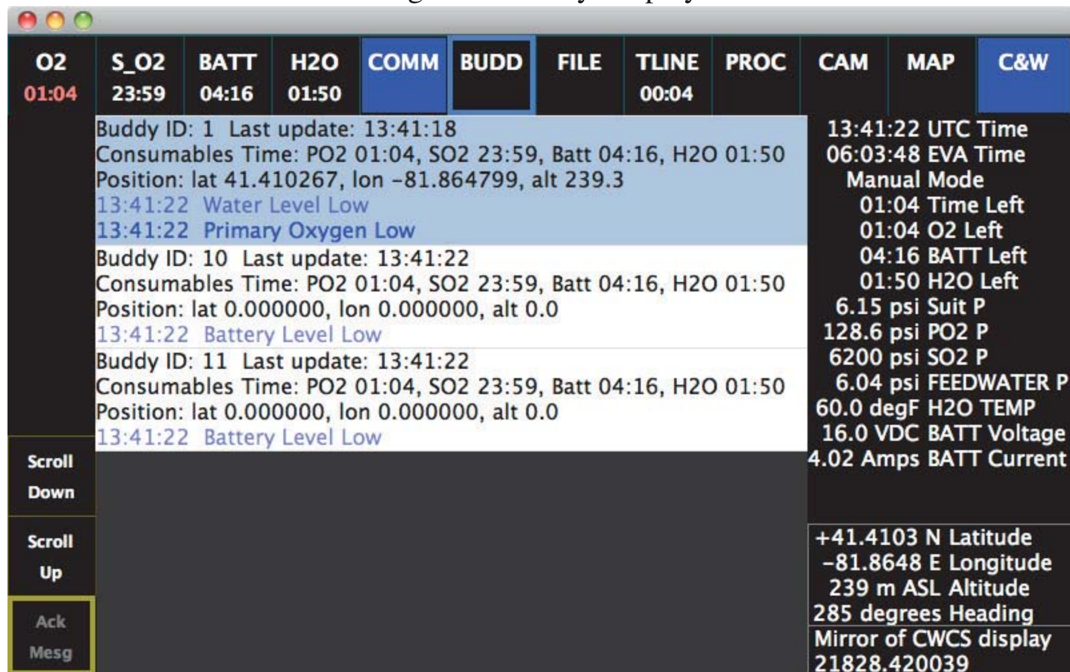


Figure 7: Buddy Display



indicator on every screen that a new message has arrived. A new message also triggers a network message to the Audio subassembly to play an *alert* sound.

The menu items on the left side of the display can be used to scroll up and down the list, and the *Ack Msg* menu item will acknowledge that current messages have been viewed by turing off the blue highlight on the COMM tab.

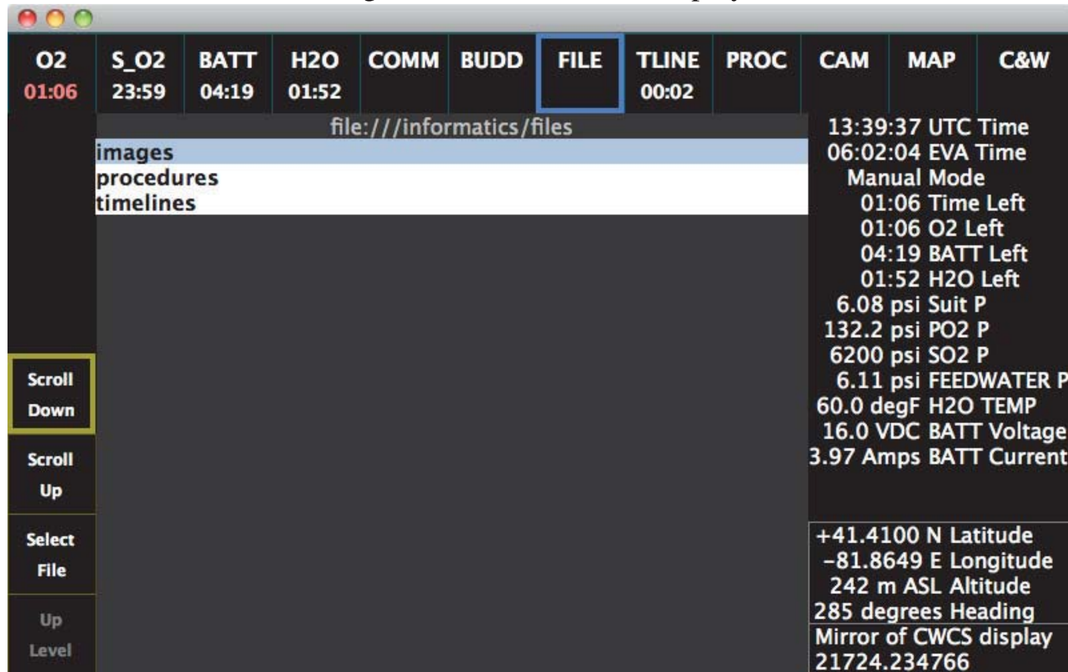
The message list has a fixed upper limit to its size (currently 50) to limit memory consumption of the program. If more than the maximum number of text message are received, the oldest ones will be dropped form the list.

4.2.3 Buddy Display

The Buddy Display (shown in Figure 7) shows information from other EVA assets. This information is currently generated by the Informatics software, and for testing purposes the Buddy Display has been modified to display information sent from itself (which would normally be ignored).

The buddy information displayed includes a suit ID number, the UTC time of the last update from a suit, time remaining for the four consumables, position (latitude, longitude and altitude) reported by the other suit, and a line for each caution or warning message generated on the other suit. Receiving a new caution or warning message also triggers a network message to the Audio subassembly to play an alert sound and highlights the BUDD tab at the top of the screen. The *Ack* button acknowledges the caution or warning and removes the highlight.

Figure 8: File Selection Display



4.2.4 File Selection

The Informatics subassembly has a file system that holds images, procedures, and timeline files that the astronaut can view. The File Selection display, shown in Figure 8, provides a way for the astronaut to navigate the file system and select a file for viewing.

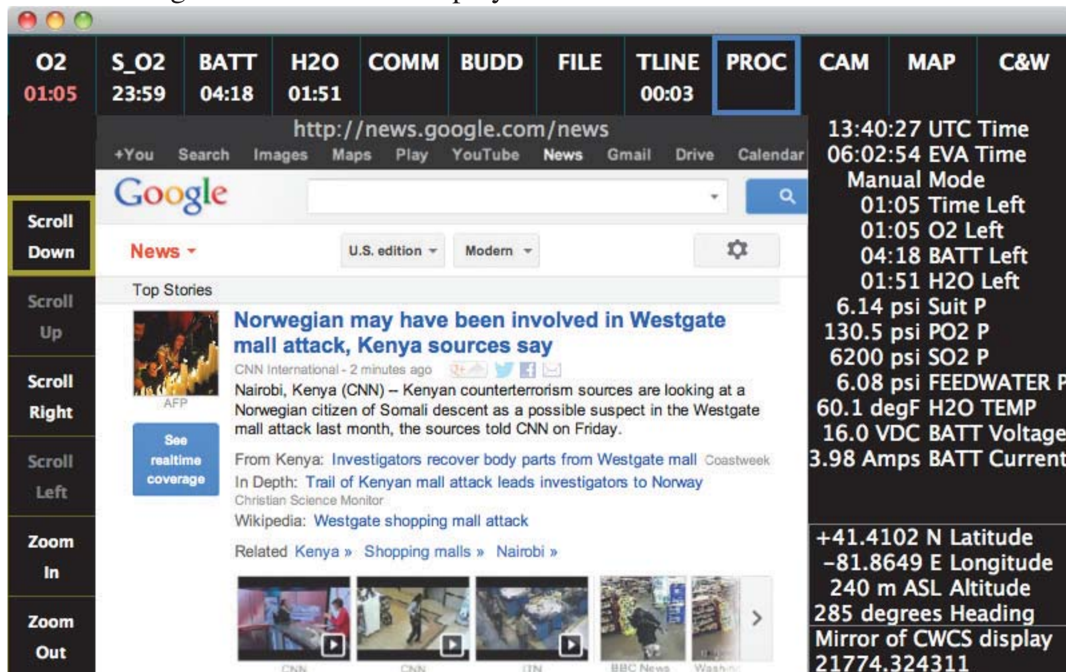
The current directory being viewed is shown at the top of the center section of the display, and the contents of the directory are shown as a list below. Directory names are shown in bold to distinguish them from regular files. The currently selected file or directory is highlighted in blue.

The Scroll Up and Scroll Down menu items on the left of the screen change the currently selected file or directory. Choosing the Select File menu item when a directory is highlighted changes the current location to that directory and shows its contents. Choosing the Up Level menu item goes back to the parent directory. The Up Level menu item will be dimmed and disabled if the current directory is the highest level allowed (the File Display can not be used to explore the entire computer).

Choosing the Select File menu item when a file is highlighted will load the file and switch to the appropriate tab to display it. The Timeline Display (TLINE) will display timeline files, and the Procedure Display (PROC) will display images or procedures.

If a mouse or a touchscreen is available, they can be used to select files and directories.

Figure 9: Procedure Display is a full-featured HTML renderer



4.2.5 Procedure Display

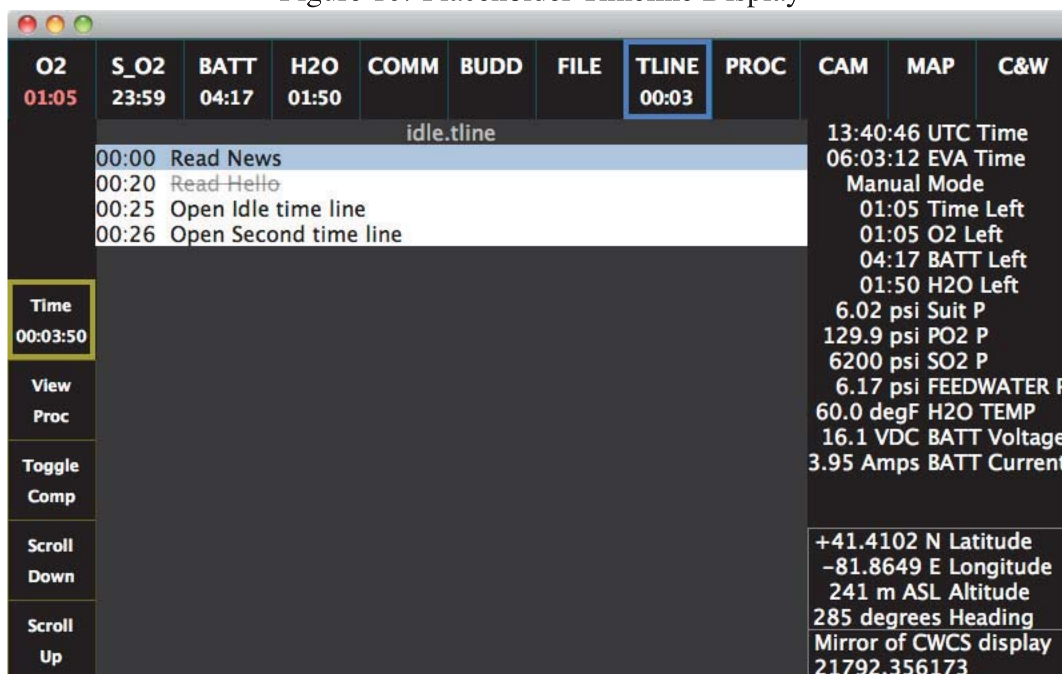
The Procedure Display is used to display images or procedures that the astronaut has selected. These images and procedure files are stored on a file system, and may be updated by ground systems or other assets using a file transfer protocol. Images may also come from an Informatics camera (once that functionality is implemented).

Procedure files are formatted as standard HTML, and may contain text markup, links and images. The Procedure Display has the features of a web browser. In fact, it is not restricted to loading data from the local file system, and if it has access to a larger network, it can load data from standard HTTP URLs. Figure 9 demonstrates this by rendering an Internet news site. There is no interface for entering an HTTP URL, but a procedure file can be loaded that contains an HTTP redirect command. Actual procedures that an astronaut might perform have yet to be written, but there are a few placeholder files for demonstration purposes.

The menu items on the left of the screen can be used to pan and zoom around the displayed image or procedure. If a mouse or a touchscreen in available, they can be used to select and follow HTTP links. There is currently no interface for selecting an following a link via the keyboard or pushbuttons.

The Procedure Display (and other displays) stores its current position within a page, so the astronaut can switch to another tab to check something and then switch back, returning to the same place.

Figure 10: Placeholder Timeline Display



4.2.6 Timeline Display

Timelines are lists of procedures with expected starting times and completion status indications. They serve as a "to do" list for astronaut activities.

The Timeline Display is shown in Figure 10. The contents of a particular timeline are loaded from a file on the file system. The contents of the placeholder file used for testing that resulted in Figure 10 are shown in Listing 4 to demonstrate the file format.

Each line in the timeline display list begins with a starting time (in hours and minutes) relative to the start of the list. The first entry will will always be 00:00. In the example shown, the second entry has a start time of 00:20, so the first entry is expected to require a total of 20 minutes. The third entry has a start time of 00:25, so the second entry is expected to take 5 minutes.

After the starting time, the timeline display list has a short title. If the activity is marked as completed, the title will be dimmed and a strike-through font will be used (as demonstrated in the second entry).

A timer showing the current number of hours and minutes since a timeline was loaded is shown in the tab display underneath TLINE. The same timer (but including seconds) is shown in the menu on the left of the screen. The timer can be paused or continued by selecting the time display in the menu with the keyboard, pushbutton, mouse, or touchscreen.

Additional menu items allow the astronaut to toggle the completion status of an entry, scroll up and down the list of entries, and view the procedure associated with an entry. If a procedure is viewed,

it will be loaded and the Informatics display will switch to the PROC tab. It is also possible to have another timeline as an entry. If a timeline entry is viewed, it will replace the currently shown timeline and reset the timer to zero,

The timeline files are structured using the JSON text format demonstrated in Listing 4. For each list entry, a starting time, completion status, title, and path are required. The path is the file system path to the new procedure or new timeline that will be loaded if the entry is selected. These timelines files are read-only for the GUI, so the files are not changed if the completion status of an item is toggled.

Listing 4: Placeholder timeline file: *idle.tline*

```
[{"start":0,  "complete":false, "title":"Read News",              "path":"/informatics/files/procedures/GoogleNews.html"},
 {"start":20, "complete":true,  "title":"Read Hello",             "path":"/informatics/files/procedures/helloworld.html"},
 {"start":25, "complete":false, "title":"Open Idle time line",    "path":"/informatics/files/timelines/idle.tline"},
 {"start":26, "complete":false, "title":"Open Second time line", "path":"/informatics/files/timelines/second.tline"}
]
```

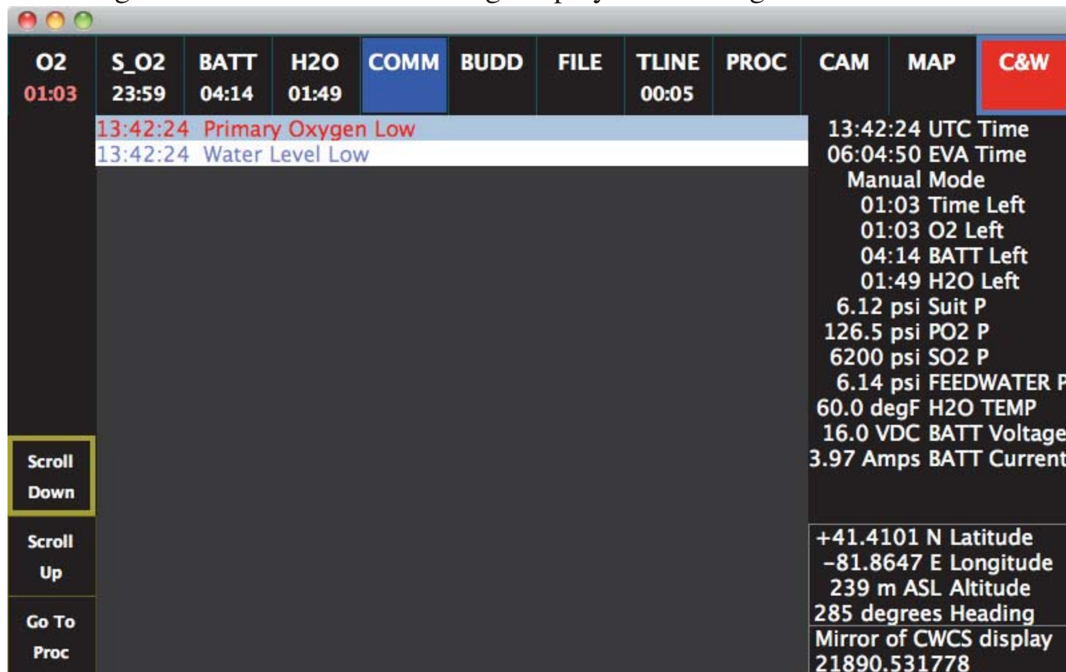
4.2.7 Caution and Warning Display

The Caution and Warning Display records caution and warning messages in a list, as shown in Figure 11. The list (and display) is initially empty, but one entry is created when a new unique caution or warning message is received (for example, *Battery Level Low*). If more messages arrive for the same condition (*Battery Level Low*), the data for the existing entry in the list will be updated. Updating existing entries is preferred over adding new entries to prevent the list from becoming huge and to give the astronaut the ability to select an entry (which would be difficult if the list is quickly changing order or scrolling).

The first item in an entry is the timestamp associated with the caution or warning. It is followed by a short readable description of the condition. The color of the entry indicates whether it represents a caution (blue) or a warning (red). Entries may have a status of either *acknowledged* or *not acknowledged*. Acknowledged entries are faded to be less noticable. If any entry is not acknowledged, the C&W tab will change color to show that it needs attention (red if there are any un-acknowledged warnings, otherwise blue if there are any un-acknowledged cautions). The Informatics Caution and Warning Display is only a display (output only), so it has no interface for changing the acknowledged state of a a caution or warning.

The menu items on the left side of the display can be used to scroll through the list or to load a procedure describing what to do about the caution or warning. If such a procedure does not exist, the *Go To Proc* menu item will be dim and have no effect. If a procedure is selected, it will be loaded from the file system and the Informatics display will switch to the PROC tab.

Figure 11: Caution and Warning Display announcing low consumables



5 Summary

An Informatics prototype for space suit graphical displays was developed. Several open source software libraries were leveraged to ease development, reducing the schedule and budget problems often associated with creating graphical software. The Qt graphics library allowed creating the displays primarily using a declarative markup language, which is much easier than using a C/C++ language application programming interface.

The ZeroMQ library hid the details of implementing a publish/subscribe network architecture, and the Nanopb implmentation of Google Protocol Buffers made creating and updating the network message formats easy. Good software libraries and inexpensive but full featured development hardware allowed one person to develop and test the entire prototype in under 6 months.

6 References

The Informatics software makes use of several open source software libraries. These libraries are primarily documented at their web sites.

More information about the *Qt* graphical user interface library is here:
`http://qt-project.org`

More information about the *ZeroMQ* network library is here:
`http://www.zeromq.org`

More information about the *Google Protocol Buffers* network wire format is here:
`http://developers.google.com/protocol-buffers`

More information about the *Nanopb* C language interface for protocol buffers is here:
`http://koti.kapsi.fi/jpa/nanopb`