AN XML-DRIVEN ARCHITECTURE FOR INSTRUMENTATION COCKPIT DISPLAY SYSTEMS

Michael Portnoy
Software Development Engineer
Teletronics Technology Corporation
Newtown, PA USA

Albert Berdugo
VP of Advanced Product Development
Teletronics Technology Corporation
Newtown, PA USA

ABSTRACT

Designing and implementing an instrumentation cockpit display system presents many unique challenges. The system must be easy to use, yet highly customizable. Typically, these systems require an experienced programmer to create graphical display screens. Furthermore, most current display systems do not provide for bi-directional communication between the instrumentation system and the display system.

This paper discusses an architecture that addresses these issues and other common problems with cockpit displays. This system captures data from the instrumentation system, displays parameters, and returns calculated parameters and status information regarding pilot actions to the instrumentation system. Unlike traditional systems, the configuration of the graphical presentation of the cockpit display can be done by a non-programmer. All communication between the instrumentation system and the cockpit display system is done transparently using XML. The usage of XML in this system facilitates real-time form previewing, cross-platform compatibility, and seamless transitions between project management, graphical configuration, and engineering unit conversions.

KEY WORDS

Cockpit Display System, XML, On-Board Processing, Embedded Systems

INTRODUCTION

The task of designing and implementing an instrumentation cockpit display system is a complicated and challenging task because a careful balance must be struck between usability and functionality. This paper provides a discussion of one approach to designing an instrumentation cockpit display system that addresses both of these issues. This system utilizes three separate hardware components along with a variety of software tools to provide flight test engineers with a complete solution for configuring and programming the cockpit display system. Using the provided tools, the user can configure the graphical presentation of the cockpit display system, configure engineering unit conversions, and can reinsert processed data into the PCM stream.

Following a brief introduction to the hardware components, the system is described in detail with an emphasis on the overall architecture, design decisions that were made, and their impact to the user and overall system functionality. The various development tools and languages that were used have also been highlighted. Finally, the configuration software is described to illustrate the process that a typical user would go through when configuring the system.

HARDWARE BACKGROUND

This cockpit display system (CCDU-2000J) is comprised of three major functional units: the embedded processing unit (ICC-2000J), display panel unit (MFD-2000J), and switch panel control unit (CDC-2000J). Different configurations are possible but each configuration always includes the embedded processing unit and display panel unit. Likewise, the functional units can share an enclosure or can be enclosed separately, which provides more capabilities. Each of these items are shown in Figure 1.

Figure 1. CCDU-2000J Cockpit Control and Display System.

INTEGRATED COCKPIT CONTROL (ICC-2000J)

The Integrated Cockpit Control (ICC-2000J) consists of a 586 PC processor running a variant of Redhat Linux 7.3. A flash memory Disk on Chip (DOC) is used to store the Linux operating system and all other files used by this system. The ICC-2000J is the control unit for the entire system and manages the flow of data between the different components. The cockpit control unit has the following key features:

- 586 PC Core Processor
- 4 Serial Ports (RS-232 / RS-422)
- 10/100 BaseT Ethernet Port
- VGA display output
- Composite Video Output
- CAIS Remote DAU function

- Frame Correlator function
- MFD-2000J and CDC-2000J Interfaces
- General Purpose I/O
- Dual function CDC Lamp Power Control

COCKPIT DISPLAY CONTROL (CDC-2000J)

Historically, switch panels have consisted of dummy switches that are used to control recorders, transmitters, and other subsystems. In this system, additional capabilities are provided by tying the Cockpit Display Control panel (CDC-2000J) directly to the embedded processing unit. For instance, data from instrumentation such as pilot status can be processed and sent back to the instrumentation system for recording, transmitting, and safety monitoring. The switch panel unit has the following key features:

- 10-switch matrix capable of Full Brightness or Night Vision Imaging System (NVIS) Dimming
- DAS Format Pushwheel
- User accessible switch contacts and LEDs
- Serial Bus connection to ICC-2000J
- CAL and Format isolated external outputs for DAS use
- RS-232 serial input port
- 4-digit 7-segment display

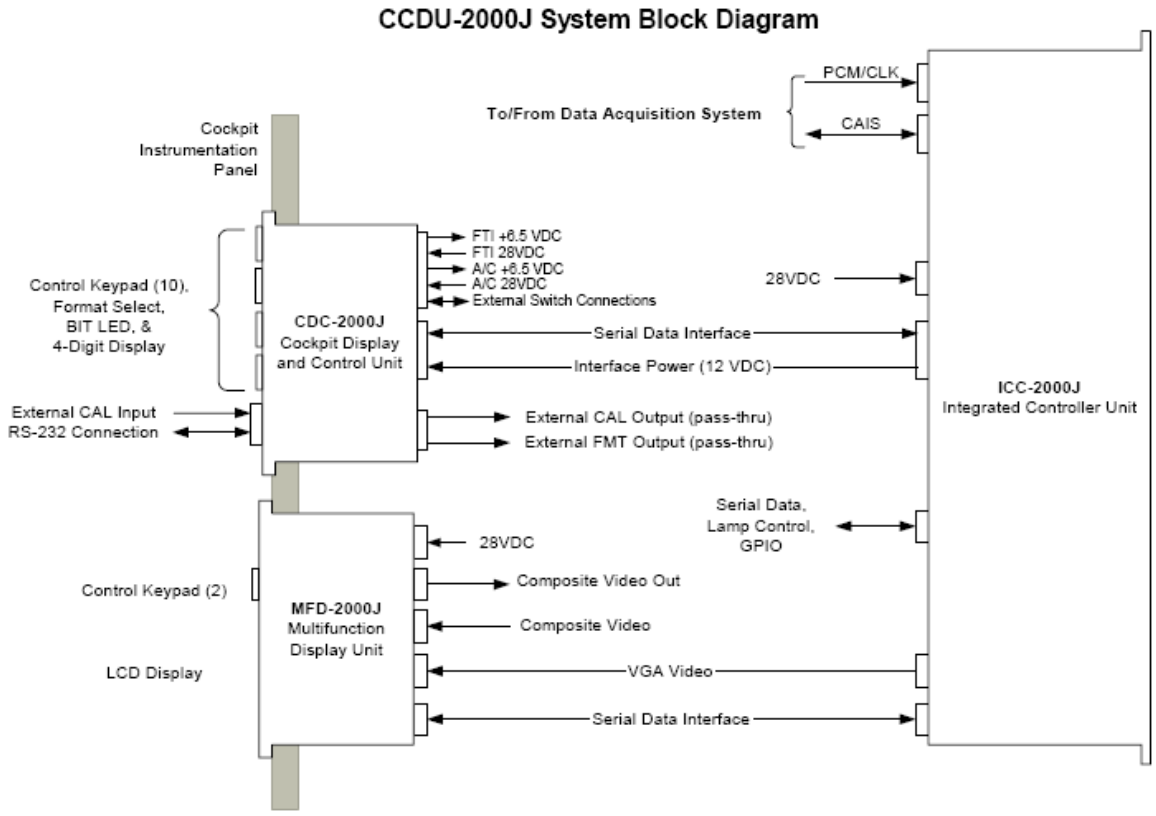
MULTI-FUNCTION DISPLAY (MFD-2000J)

The Multi-Function Display (MFD-2000J) that is used in this system can either display data via a VGA input from the embedded processing unit or function as a dual-purpose display. When used as a dual-purpose display, an onboard camera can be connected to a composite video input on the rear of the display panel to view events such as missile separation. The display panel unit has the following key features:

- 3.5 inch diagonal LCD display
- 2 soft-function push buttons
- Standard VGA signal input
- Composite Video input and output
- Serial Bus connection to ICC-2000J

Figure 2 is the CCDU-2000J system block diagram, which describes how the three components are connected.

Figure 2. CCDU-2000J System Block Diagram.



FEATURES / CAPABILITIES OF THE SYSTEM

Using this system, a user can display data graphically using widgets (LCDs, strip charts, LEDs, labels, dials, status bars, etc), configure engineering unit conversions, and reinsert processed data into the PCM stream. Reinserted data can then be viewed, analyzed, or recorded as part of the standard PCM stream. Cockpit display configuration software is provided for use in configuring and programming this system in conjunction with the system configuration software. The following engineering unit conversions (EUCs) are supported:

- Unprocessed Data
- Polynomials (up to 10th order)
- Interpolations / Lookup tables (up to 32 input-output pairs)
- IRIG Time Processing
- Average, Minimum, Maximum
- Derived calculations using a calculator-style interface

After configuring an EUC, the user can take the resulting processed data and reinsert it into the PCM stream. This is accomplished by specifying that an EUC is going to be outputted to the CAIS-bus, creating a new output parameter to sample the processed value, and then placing this new output parameter into the PCM stream. The processed data will then be sampled every time that the parameter is sampled in the PCM stream. This processed data can then be reprocessed if

so desired. Reprocessing data that has already had an engineering unit conversion performed on it is useful for data type conversion.

After an EUC has been configured, the resulting processed data can also be scaled. This allows the user to change the output range of an EUC to any desired range of values. For example, a raw count from 0 to 4095 could be scaled to the range 0 to 100 for display as a percentage on a progress bar. The output of an EUC can also be limit checked. If the output falls outside the specified safety limits, a graphical alarm is displayed onscreen.

SOFTWARE DESIGN ISSUES

During the early stages of development, it became clear that several key design issues would have to be resolved:

> What operating system should be used and what real-time capabilities would be required?
>
> What programming language(s) would be used to implement the software requirements of the system?
>
> What transport mechanism would be used to support data interchange within the system?
>
> How to prevent file system errors caused by unexpected power removal?

A wide range of operating systems were considered for this system including Microsoft Windows XP Embedded, several real-time Linux variants, QNX, LynxOS, and several standard Linux variants. It was determined that real-time capabilities were unnecessary since the main purpose of this system was to provide current value information to the pilot. Due to the high level of customization and control over the operating system that would be required, Linux was chosen over embedded MS Windows XP and other non-Linux operating systems. Finally, Redhat Linux 7.3 was chosen because of its rich user base, ease of use, and existing body of hardware drivers and application software.

When choosing a programming language, the main considerations were: the existence of cross-platform graphical widgets, development costs, and performance. The key issue was determined to be the existence of cross-platform graphical widgets since graphical forms that were created on MS Windows-based user computers had to look identical when uploaded to the Linux-based embedded system. As a result of this requirement, the following languages and toolkits were considered: Java, C++ with the QT application framework, and Tcl/Tk.

After much consideration, C++ with the QT application framework was selected for the following reasons. First, the QT framework was designed from the ground up to be a cross-platform GUI development toolkit. In addition, there are many graphical widget packages available on the Internet. This was essential since there was no desire to design strip charts, histograms, and more complicated widgets from scratch. However, the most important features of QT were that it has the ability to dynamically load graphical forms at runtime and that it uses XML to store its graphical forms. These two features were key ingredients to the overall system design and will be discussed in depth later on in this paper.

Finally, a platform-neutral data exchange mechanism was needed to transmit data between the MS Windows-based user computers and the Linux-based cockpit display system. XML was a strong contender since the decision to use QT had already been made. The other possibilities that were considered included a simple, flat text file and a cross-platform relational database

solution such as MySQL. XML was chosen because of its human readability, preponderance of existing tools, and because it was the native language for QT graphical forms. This allowed for a completely unified approach to the exchange of data.

Unfortunately, one limitation of the Linux operating system that had to be overcome was that that Linux file system does not respond well to the immediate removal of power. As a result of this problem, it was decided that the solid state Disk on Chip (DOC) would be divided into three separate partitions. The partition that was occupied by the Linux kernel and other core Linux files would be mounted as a read-only partition so that no data loss would occur when the system suddenly lost power. Cockpit display projects and temporary files would be stored on a writable partition to facilitate proper operation of the system. Finally, a read-only backup partition would be provided in case of system failure so that the other partitions could be reconstructed without requiring the unit to be returned for repairs.

SYSTEM PROGRAMMING

Two different methods of programming are supported: programming over the CAIS-bus using the system configuration software and over RS-232 using a client tool that communicates with a server that is running on the cockpit display system. Programming via the CAIS-bus is the easier way to program the system but this programming method is limited by the internal memory that is accessible via the CAIS-bus. The client-server programming mechanism removes this memory limitation and allows the system to be programmed directly from the front of the CDC-2000J via RS-232. When an RS-232 cable is inserted into the receptacle on the CDC-2000J, the insertion is detected by the software driver, the system switches to a programming mode, and the server is loaded. After the user has finished uploading the cockpit display configuration using the provided client tool (MS Windows-based application), the RS-232 cable is removed, and the system resumes normal operation.

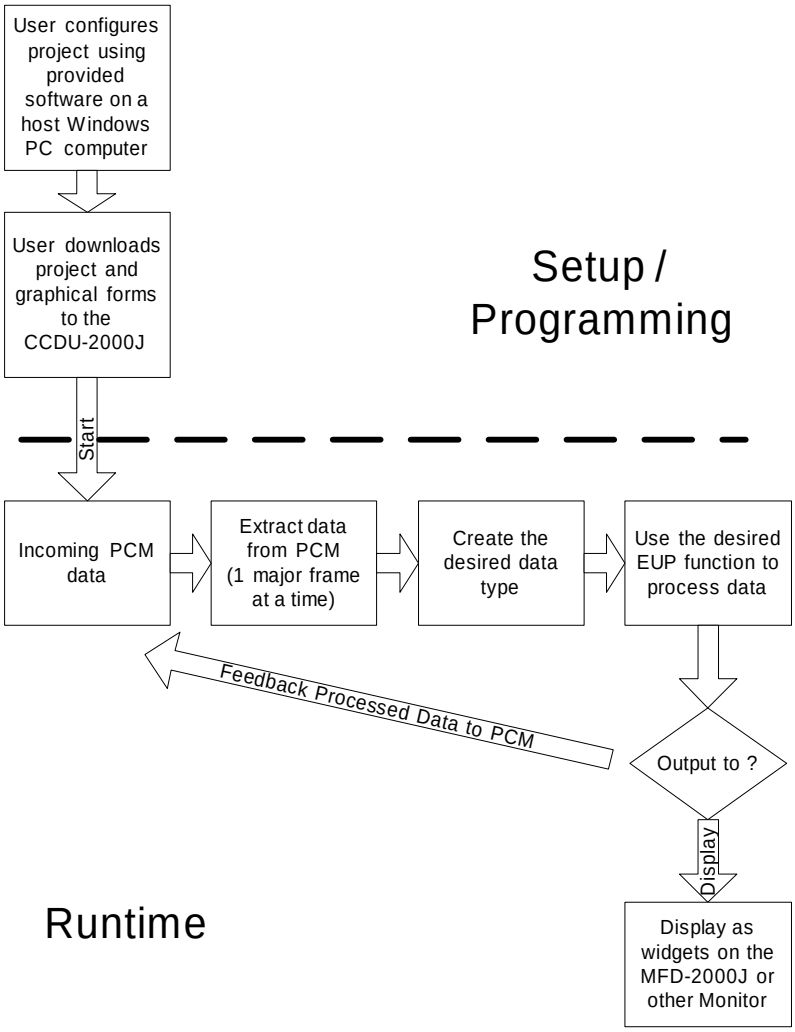
DATA FLOW

Due to the complexity of the system, configuration of the system requires switching between the system configuration software and the cockpit display setup software. First, the user must configure the data acquisition hardware and define a PCM format. Once the PCM format has been defined, the cockpit display setup software is launched. The cockpit display setup software receives the PCM format from the system configuration software automatically via XML. Next, the user configures the graphical pages and engineering unit conversions. After completing the configuration, the user is returned to the system configuration software. If the user only desires to display processed data graphically on the MFD-2000J, the hardware can be programmed immediately and the system will be ready for data acquisition.

However, if the user needs to reinsert processed data into the PCM stream, then the parameters that were created using the EPMU must first be placed in the PCM format before the hardware is programmed. The user can then program the hardware unless they would like to reprocess the processed data. If that is the case, the EPMU is reopened and process is repeated.

Once the project is uploaded to the hardware, the ICC-2000J extracts the project's XML configuration files and graphical forms. Next, the DisplayEngine program loads and the system begins processing and displaying data as shown in Figure 3.

Figure 3. Data Flow within the CCDU 2000J System.

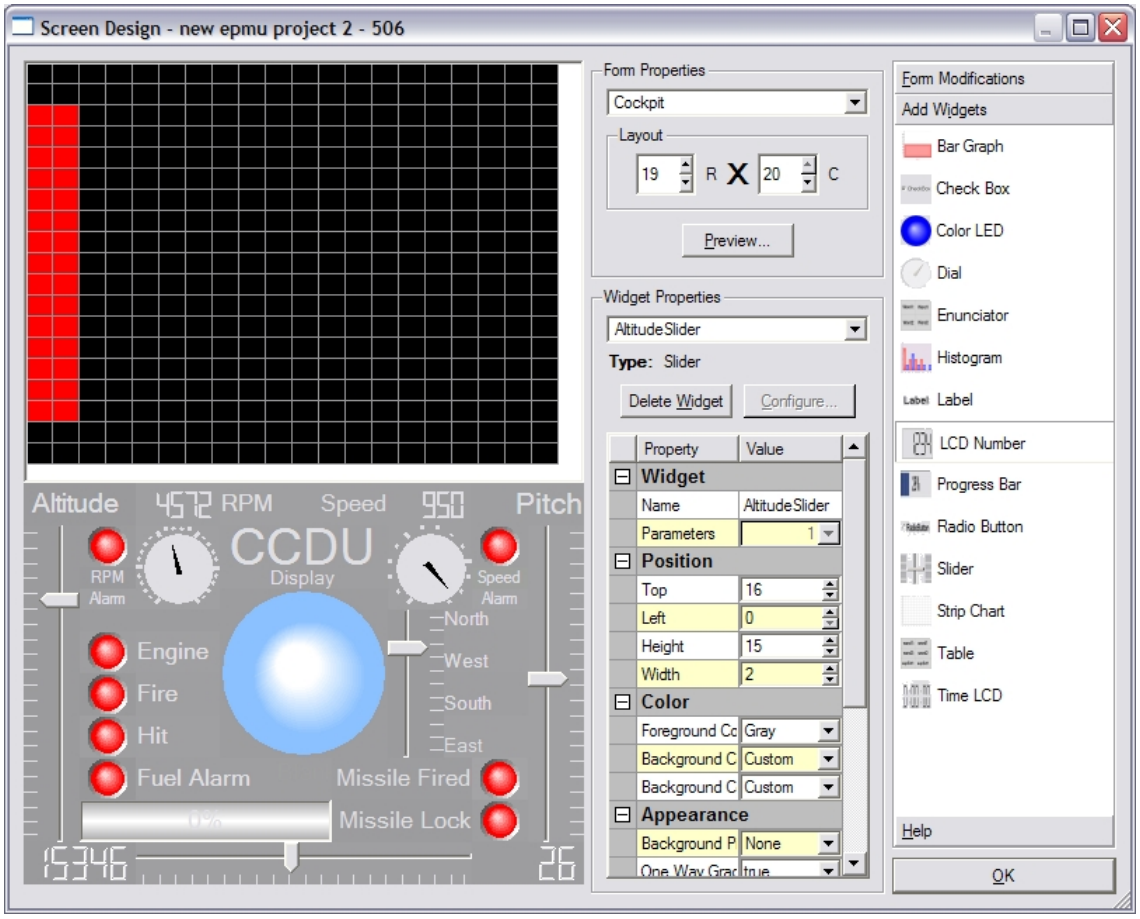


COCKPIT DISPLAY CONFIGURATION SOFTWARE (EPMU)

The cockpit display configuration software (also known as the EPMU) is used to configure all aspects of the cockpit display system. All project-wide settings are stored within a single XML configuration file, which is the foundation of each cockpit display project.

Figure 4 shows the screen used to create and lay out graphical widgets on forms.

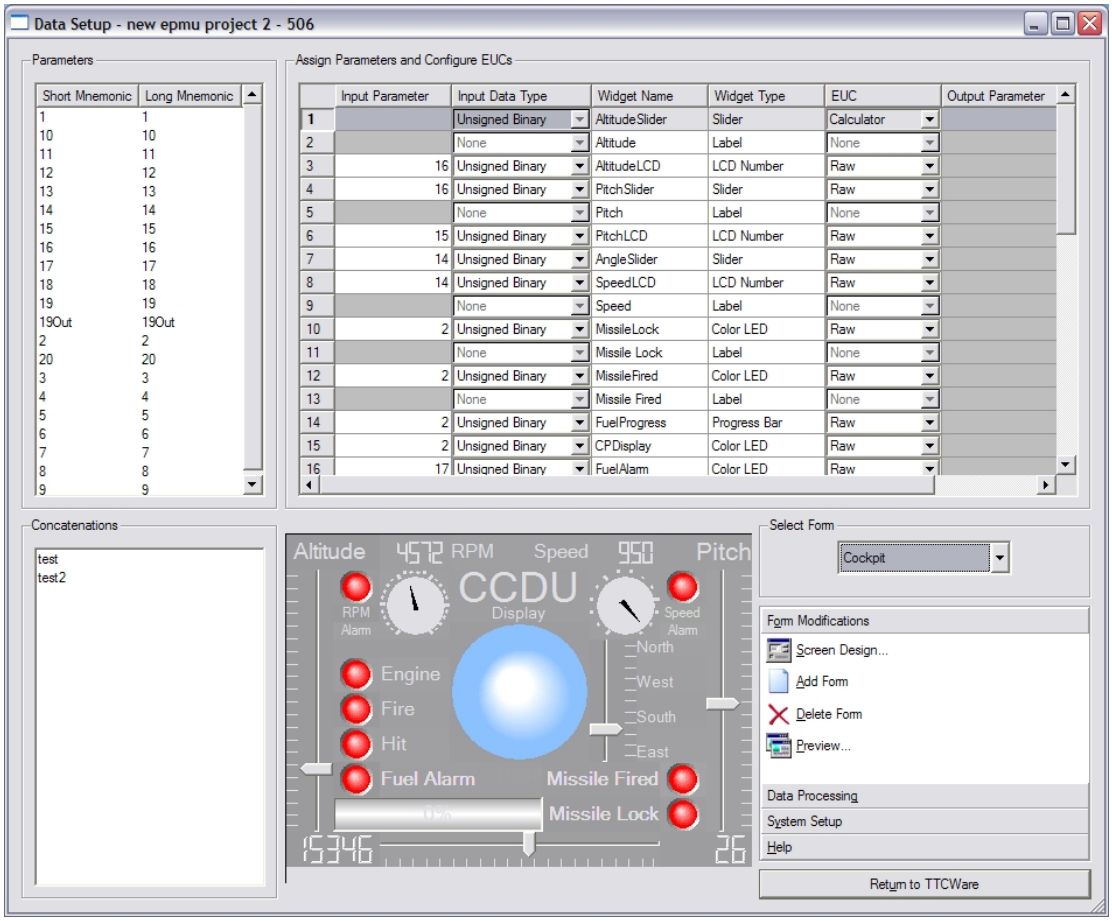
Figure 4. Screen Design.



Each form is partitioned into a grid of blocks. These blocks aid in the placement of widgets and handle widget alignment automatically. Using this screen, the user can customize the graphical properties of each widget. As a property is changed, the preview window at the lower left is updated in real-time. This is accomplished using the dynamic loading capabilities of QT. The EPMU reads and writes directly to the QT graphical forms that are stored as XML files. When a user changes a property, the change is written immediately to the XML form, which is then dynamically reloaded. This allows the system to function without needing to compile the QT forms as would normally be required.

After creating and laying out the graphical forms and widgets, the next step for the user is to bind the widgets to parameters that have been placed in the PCM format. This step is done by dragging and dropping parameters onto widget entries in the table as shown in Figure 5.

Figure 5. Data Setup Screen.



Once a parameter is bound to a widget, the user can specify the input data type and engineering unit conversion that will be applied to the parameter. Many different data types are supported including unsigned binary, IEEE 754 floating point, and binary coded decimals (BCD).

In addition to the built-in engineering unit conversions, a graphing calculator style interface is provided that allows the user to create custom EUCs involving multiple parameters. Several built-in functions are included and an interface is provided to graph and evaluate the custom EUCs.

The EPMU supports several more features that have not been previously mentioned, including the ability to create concatenations of up to four parameters. Once created, a concatenation can be displayed on graphical widgets and can be used as an input to an EUC. In addition, the user can create bit masks to select specific bits from within the parameters in a concatenation. For example, the user could display the value of a single bit from within a parameter in the PCM format on an LED. In addition, the EPMU also allows the user to customize the functions of the soft keys on the MFD-2000J and to change the page ordering of the graphical forms.

CONCLUSION

Hopefully, this paper will serve as a roadmap for future cockpit display system design. Overall, it demonstrates that one can design an instrumentation cockpit display that is simultaneously flexible, powerful, and easy to use. By combining platform independent tools such as the QT application framework and XML, one can create a system that seamlessly moves between MS Windows-based configuration and Linux-based operation. This allows the user to take advantage of existing familiarity with MS Windows-based PCs without sacrificing the power and customization provided by the Linux platform. Furthermore, this system demonstrates that instrumentation cockpit display system can be an invaluable tool for data acquisition and a valuable addition to any telemetry system.

REFERENCES

1. Dalheimer, Matthias Kalle. Programming with QT, Second Edition. Germany: O'Reilly, 2002.
2. Portnoy, Michael. "An Implementation of a Complete XML System for Telemetry System Configuration", ITC Proceedings, Volume XXXXI.
3. The homepage for QT is: http://www.trolltech.com/products/qt/index.html