# Enhancing International Space Station (ISS) Mission Control Center (MCC) Operations Using Tcl/Tk

Brian O'Hagan
NASA/Johnson Space Center
brian.ohagan-1@nasa.gov

Stephen K. Long, Sr.
United Space Alliance
stephen.k.long1@jsc.nasa.gov

## Abstract

This paper will discuss the use of Tcl/Tk to enhance the abilities of flight controllers to control the International Space Station (ISS) from the Mission Control Center (MCC) at the Johnson Space Center. We will discuss why existing tools where not able to meet these needs as easily as Tcl/Tk. In addition, we will also discuss how we interfaced with the existing MCC infrastructure to receive ISS telemetry, find servers, register services, and send commands to ISS.

## Extended Abstract

Existing MCC applications where developed for use on the X11 platform. These applications where typically coded in either C or C++. This resulted in the typical development limitations were changes where costly, time consuming, and difficult to implement. Several of these applications make use of Meta data files to allow easy definition of telemetry, commands, and computations though most only support one of these types. The application that came closest to supporting all of these functions was part of the Common Display Development Team (CDDT). This application is used by both the crew on ISS and flight controllers in the MCC. This application is coded in C++ and combines the use of telemetry, computations, and commands on the same displays using XML data files. The main limitation of this application is its dependency on the ISS software configuration, which limits the delivery of updated data files to once or twice a year. Additionally, the application has limited graphical capabilities and a large backlog of deficiency. We will discuss how we addressed these limitations using Tcl/Tk.

The Information Sharing Protocol (ISP) is the protocol used to pass telemetry, computations, and status data between workstations in the MCC. ISP uses the standard client server architecture and allows data to be multicast on a change-only basis unlike telemetry that downlinks all data all the time. ISP is coded in C and uses a library of APIs for accessing its functionality. We will discuss how we where able to interface Tcl/Tk applications with the ISP APIs to access telemetry data, register callbacks, commands for telemetry events, and provide for multiple server connectivity. We will also discuss telemetry data management, callback management, performance tuning, and publishing of data to other ISP clients.

Using the ISP interface, Tcl/Tk applications were developed for Caution and Warning management, external camera control, automated procedure management on ISS, and for displaying telemetry. These applications were developed in a shorter period of time, with fewer defects, and with easier to use graphical user interfaces while maintaining the existing rigorous Configuration Management and certification standards prior to their use during ISS operations.

The MCC is a distributed architecture that allows for user access throughout the MCC. Since users can log into any workstation to support a particular activity and ISP Servers run on each users workstation, the typical approach of finding a server by connecting to an IP address cannot be used. Instead, the MCC uses Network Registration Services (NRS) which defines a list of service IDs with the associated workstation and port. Applications then use NRS to lookup a service ID to determine the workstation and port to connect to. We will discuss how Tcl was interfaced to NRS to allow for registering service IDs and for finding services. This allowed the Tcl/Tk application Caution and Warning Status Tracker and Analyzer (C&W STAN), to provide server functionality using HTTP in a manner like web services for ISS Caution and Warning event management.

We will also discuss how we were able to interface Tcl with the ISS Command Server for sending commands to ISS and for receiving command related event updates. Like ISP, the Command System is coded in C and C++, and uses a library of APIs for accessing its functionality. Existing MCC applications were limited to only sending commands and receiving a command accepted or rejected response. They did not have the capability to verify a command executed properly with end item telemetry. This required operator intervention for all command verification steps. We plan to discuss how the commanding interface allowed us to automate routine operations, provide an easier interface for camera control, and an enhanced scripting capability. We will also discuss command data management, callback management, and performance tuning.