

**TREK**

**HOW TO PROVIDE ADDITIONAL  
COMMAND VALIDATION WHEN USING  
COMMAND MANAGEMENT**

**TUTORIAL**



**November 2012**

Approved for Public Release; Distribution is Unlimited.

## TABLE OF CONTENTS

<u>PARAGRAPH</u>	<u>PAGE</u>
<b>1 What You Need to Know Before You Read This Document .....</b>	<b>1</b>
<b>2 Technical Support.....</b>	<b>1</b>
<b>3 Introduction.....</b>	<b>2</b>
<b>4 Writing Command Validation Software for TReK .....</b>	<b>3</b>
4.1 First Steps: Setting Up Your Development Environment .....	3
4.2 Knowing When You Need to Validate a Command .....	3
4.3 Validating the Command and Sending Back the Results .....	4
4.3.1 <i>Getting the Command from TReK</i> .....	4
4.3.2 <i>Sending Back the Results</i> .....	5
4.3.3 <i>A Short Code Sample</i> .....	5
4.4 Putting It All Together .....	6
<b>5 Command Management Tutorial with Validation .....</b>	<b>8</b>
5.1 Setting Up the Command Node for User Provided Validation .....	8
5.2 Sending a Command from the Sub Node.....	9
<b>Appendix A Command Validation Source Code Snippet .....</b>	<b>16</b>
<b>Appendix B Glossary .....</b>	<b>19</b>
<b>Appendix C Acronyms .....</b>	<b>26</b>

## FIGURES

<u>FIGURE</u>	<u>PAGE</u>
Figure 1 Advanced Management Dialog .....	8
Figure 2 Advanced Management Dialog with Validation Checking On .....	9
Figure 3 Command Processing Main Window with TRR Column .....	10
Figure 4 Modify Command Dialog .....	11
Figure 5 Modify Command Field Dialog .....	12
Figure 6 Command Track (Validation Timeout) .....	13
Figure 7 Realtime Commanding Messages (TRR Error 5) .....	14
Figure 8 Command Track (Success!!!) .....	15

CODE SAMPLES

<u>CODE SAMPLE</u>	<u>PAGE</u>
Code Sample 1 The Validation Event Name .....	4
Code Sample 2 Validating the Command .....	6
Code Sample 3 Simple Code Outline .....	7

## 1 What You Need to Know Before You Read This Document

This tutorial is written with the following assumptions:

- You are familiar with the material in the TReK Getting Started User Guide (TREK-USER-001), TReK Command Tutorial (TREK-USER-020) and the TReK Command Management Tutorial (TREK-USER-021).
- You are an average C or C++ programmer and are familiar with your development environment. A Visual C++ 6.0 project containing the source code described in this document is available in the TReK installation directory. This tutorial will not cover how to use the TReK Command Application Programming Interface with Visual C++. If you would like to read a tutorial that covers this for the TReK Telemetry API (it will be very similar for commanding), you can read one of the other tutorials that are provided. A good example is the How to Build a Visual C++ Computation Tutorial.

If you are uncomfortable with any of the items listed above, some of the terminology and concepts presented in this tutorial may be difficult to understand.

## 2 Technical Support

If you are having trouble installing the TReK software or using any of the TReK software applications, please try the following suggestions:

Read the appropriate material in the manual and/or on-line help.

Ensure that you are correctly following all instructions.

Checkout the TReK Web site at <http://trek.msfc.nasa.gov/> for Frequently Asked Questions.

If you are still unable to resolve your difficulty, please contact us for technical assistance:

TReK Help Desk E-Mail, Phone & Fax:

E-Mail:	trek.help@nasa.gov
Telephone:	256-544-3521 (8:00 a.m. - 4:30 p.m. Central Time)
Fax:	256-544-9353

TReK Help Desk hours are 8:00 a.m. – 4:30 p.m. Central Time Monday through Friday. If you call the TReK Help Desk and you get a recording please leave a message and someone will return your call. E-mail is the preferred contact method for help. The e-

mail message is automatically forwarded to the TReK developers and helps cut the response time.

### **3 Introduction**

This tutorial will cover three functions provided by the TReK Command API that allow a user to provide validation software in addition to validation performed by the TReK Command Processing application when a TReK system is configured as a command node. The first of these functions is `GetValidationEventName`.

`GetValidationEventName` returns a string that can be used to wait on a validation signal that is sent by TReK when an uplink request needs validating. The `GetUplinkPacketForValidation` function retrieves the uplink packet along with other information for the validation software. Once the validation is finished, the `SendValidationResults` function is used to let TReK know if validation succeeded or not.

After the details of these functions are discussed, you will be able to use the Command Management Tutorial along with an addendum provided in this document to use the user provided validation software describe in the tutorial.

## 4 Writing Command Validation Software for TReK

The Command Processing application in TReK provides some validation of commands when a destination is configured to allow sub nodes to connect and send commands. For example, the command node will check to ensure that the user is enabled and is allowed to use the command mnemonic. However, it is possible that each payload team may have additional validation requirements that are not implemented as part of the TReK Command Processing application. By configuring the destination to have user provided validation, a payload team can write additional software that provides payload specific validation of commands. The software can be written with any development tool that supports ANSI C. You can write a graphical program if you wish. However, the example described in this document will be a simple console program to allow you to easily see the interaction of the TReK Command API functions.

### 4.1 First Steps: Setting Up Your Development Environment

Before you can use the TReK Command API, you will need to copy a few files from the TReK installation to your program or project folder. Alternatively, you can point to the TReK files in their current location. You will need the TReK headers files `trek.h`, `trek_error.h`, and `trek_cmd_user_api.h` located in the include directory under the TReK installation folder. You will also need the `trek_cmd_user_api.lib` file from the library directory in the TReK installation folder.

Once you have included these files in your development environment, you are ready to start writing the command validation program.

### 4.2 Knowing When You Need to Validate a Command

If you have ever written a program with the TReK Telemetry API's `GetPacketArrivalEventName` function, then this part of the tutorial will look very familiar. When you turn on the user provided validation, TReK will generate a windows event to signal that a command needs to be validated. Your program will need to catch the event, validate the command, and send back the results. If for some reason your program does not catch the event (e.g., the program wasn't running at the time), TReK will stop waiting for the results after a timeout period and return that the command was rejected. You will need to call the `GetValidationEventName` function and make the appropriate calls to the Windows API to setup to wait on the event. The small code segment that follows uses the WIN32 API function `CreateEvent` function to create or open the event used for the signal. This will allow you to start the validation program prior to activating the destination. It is important that you use the function parameters shown for the `CreateEvent` function to ensure the function will work correctly with TReK.

```

// Don't forget to include the trek_cmd_user_api.h file in this source
// file

// variables used in example code
int return_value;
char *event_name_ptr = NULL;
HANDLE validation_event_handle;

// Get the event name from TReK for a destination named "POIC"
return_value = GetValidationEventName( "POIC", &event_name_ptr );

if( return_value != SUCCESS )
{
    // Perform error checking and exit since the program will not get
    // a signal.
}

// Create/Open the event
validation_event_handle = CreateEvent( NULL,
                                       FALSE,
                                       FALSE,
                                       event_name_ptr );

if( validation_event_handle == NULL )
{
    // Perform error checking. Could not create event.
}

// The following code will clean up the memory and resources created
// above. It should be part of the exit code of your program (e.g., at
// the end of the main function block). Do not call the CloseHandle
// function until you no longer need to wait on the validation event or
// you will introduce a bug.

delete event_name_ptr;
CloseHandle( validation_event_handle );

// End of example code.

```

**Code Sample 1 The Validation Event Name**

### 4.3 Validating the Command and Sending Back the Results

The code in the previous section contains all of the preliminary work that is needed before you can actually start validating commands. Now we can cover actually getting the command that needs validation and sending the results of the validation back to TReK so the command can be processed or rejected.

#### 4.3.1 Getting the Command from TReK

To get the command from TReK to validate, you call the `GetUplinkPacketForValidation` function. This function will return four items that you can use to determine whether or not the command should be accepted and sent on to the destination, or rejected and all processing of the command stopped. The information returned is:

- Username – The username associated with the TReK login from the sub node. If Remote Services is configured to not require a login, then the username value will be set to “no login”.

- Mnemonic – The mnemonic, or command name, being requested. TReK will perform checks to ensure that the username associated with the mnemonic is allowed before calling the user validation program. You will probably use these two items to check the command data.
- Command – The command returned includes the header, command data, and checksum. TReK will be resetting the timestamp in the header and recalculating the checksum before forwarding this command to the destination if it passes validation. Validation performed at this point could include checking to see if the user is allowed to set certain values in this mnemonic.
- Command Length – The length of the command is returned in bytes. This length includes the header, command data, and checksum.

Don't forget that TReK creates memory that should be freed by the user program for the username, mnemonic, and command. You can read the online version of the Command API Reference Manual to see more details about the `GetUplinkPacketForValidation` function.

#### 4.3.2 Sending Back the Results

Once the user program has finished validating the command, the results can be sent back to TReK via the `SendValidationResults` function in the TReK Command API. If the validation was successful, TReK will continue to process the command by forwarding it to the destination. In addition, a TReK Receipt Response (TRR) message will be generated and sent to the sub node to inform it that the command was accepted and forwarded by the command node. The sub node user can see this message in the command track area of the Command Processing main window.

If the command does not pass validation, TReK will stop processing the command and generate a TRR message that indicates that user provided validation failed. This will appear as TRR error number 5 on the sub node. The error number you send to TReK in the `SendValidationResults` function will be contained in the TRR data sent to the sub node. While the command track will not show the detailed error number, a sub node user could see the information in the Realtime Commanding Messages Viewer. It is recommended that the user program for validation be written so that the error codes returned via the `SendValidationResults` function are meaningful and can be used to determine exactly why validation failed.

#### 4.3.3 A Short Code Sample

Now that we have covered the `GetUplinkPacketForValidation` and `SendValidationResults` functions, it is time to show a simple code example on how to use these functions.

```

// Don't forget to include the trek_cmd_user_api.h file in this source
// file

// variables used in example code
int return_value;
char *username_ptr = NULL;
char *mnemonic_ptr = NULL;
unsigned char *cmd_data_ptr = NULL;
unsigned int cmd_data_length;
unsigned short validation_results;

// Get the uplink packet for validation for the destination "POIC"
return_value = GetUplinkPacketForValidation( "POIC",
                                           &username_ptr,
                                           &mnemonic_ptr,
                                           &cmd_data_ptr,
                                           &cmd_data_length );

if( return_value != SUCCESS )
{
    // Perform error checking.
}

// Validate the command. This is just a simple check. This will be
// the section of code that is payload unique.

if( cmd_data_ptr[0] == 0x01 ) // checking the first byte in header
{
    validation_results = 45;    // payload unique error
}
else
{
    validation_results = 1;    // successful validation
}

// Once the command is validated, send the results back to TReK
return_value = SendValidationResults( "POIC", validation_results );

// The following code will clean up the memory create by TReK in the
// GetUplinkPacketForValidation function.

delete username_ptr;
delete mnemonic_ptr;
delete cmd_data_ptr;

// End of example code.

```

**Code Sample 2 Validating the Command**

#### 4.4 Putting It All Together

Now that you know about the three functions provided for user validation, we can go over an example of using all the functions in a program. You will probably want your command validation software to run continuously. Otherwise, you might miss a validation signal and the command will get rejected since TReK does not receive results back from your program. The code sample that follows is a simple outline that contains a few lines of code along with comments. This code structure is the same as is used in the Visual C++ 6.0 Project available in the TReK installation directory. The project can be found in the following directory:

<TReK Install Path>\Examples\Visual C++\CommandValidation

The code sample is not meant to compile. You should consider it pseudo-code.

```
// Don't forget to include the trek_cmd_user_api.h file in this source
// file.

// Get the Validation Event Name for the destination (Code Sample 1).

// Validation loop that runs until an exit condition occurs
loop = TRUE;
while(loop)
{
    // Wait on the event using the WIN32 API.
    code = WaitForSingleObject( validation_event_handle, INFINITE );

    if( code != WAIT_OBJECT_0 )
    {
        // some error occurred, exit program.
        loop = FALSE;
    }
    else
    {
        // Validate the command (Code Sample 2)

        // Send the results to TReK (Code Sample 2)

        // Clean up memory, ready for next command (Code Sample 2)
    }
}

// Clean up any resources and exit (Code Sample 1)

// End of code outline.
```

### Code Sample 3 Simple Code Outline

## 5 Command Management Tutorial with Validation

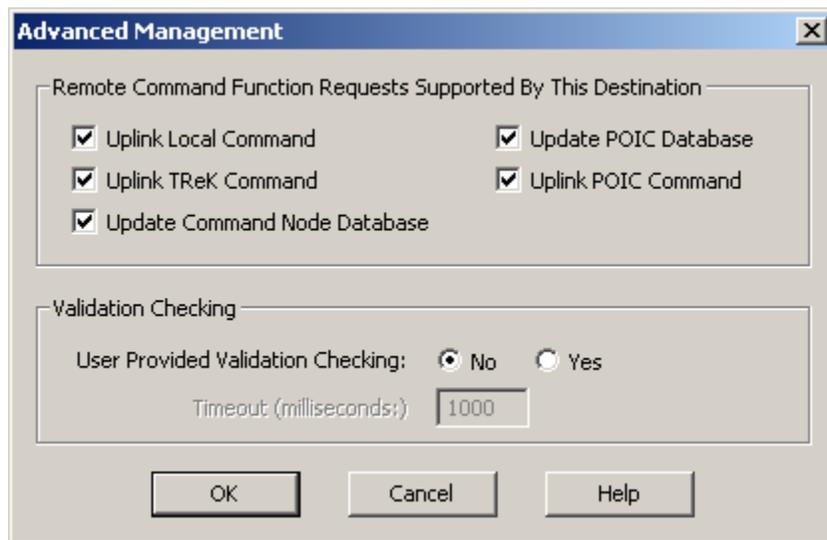
This section will use the example code that is delivered as part of the TReK examples along with the Command Management Tutorial to show a simple example of the user provided command validation. You will probably want to print out the Command Management Tutorial and this section of the document. This section will contain some changes that you will need to perform in the Command Management Tutorial in order to set up TReK for user provided validation. The changes described here are:

- Replace step number 25 in section 5.2 of the Command Management Tutorial. The changes are described in Section 5.1 of this tutorial.
- Replace section 5.5 of the Command Management Tutorial with Section 5.2 of this tutorial.

### 5.1 Setting Up the Command Node for User Provided Validation

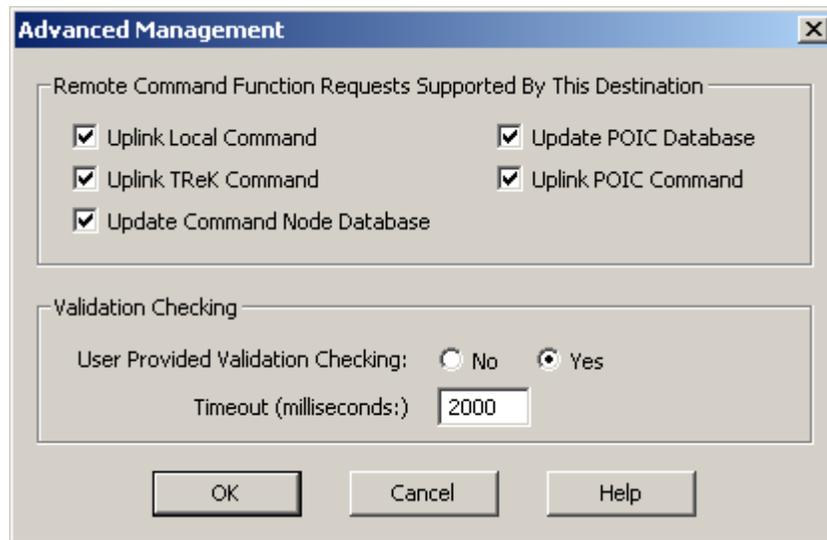
This section replaces step number 25 in section 5.2 of the Command Management Tutorial.

Push the **Advanced** button. The Advanced Management dialog shown in Figure 1 will be displayed.



**Figure 1 Advanced Management Dialog**

The Advanced Management dialog is used to configure specific management properties associated with this destination. These properties will be used for all sub nodes (remote users) that connect to the destination. As you can see, you can set the command functions that will be supported for remote users and configure user provided validation checking. Turn on user provided validation checking and set the timeout value to be 2000 milliseconds. Your dialog should now look like Figure 2.



**Figure 2 Advanced Management Dialog with Validation Checking On**

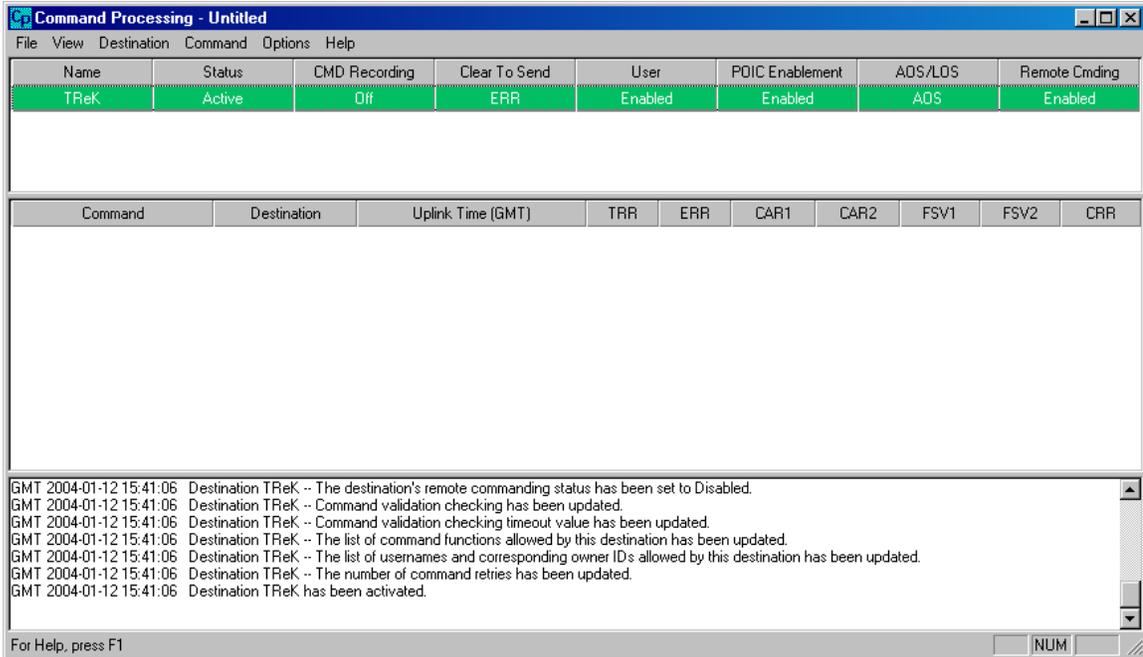
Return to Step 26 in Section 5.2 of the Command Management Tutorial.

## 5.2 Sending a Command from the Sub Node

This section replaces Section 5.5 of the Command Management Tutorial.

### Step-By-Step

1. If you haven't moved over to your sub node computer it's time to head that way. Before we send the command there's another feature in Command Processing that is helpful if you're a sub node. Go to the **View** menu and select **Set Main Window Command Track Preferences**. In this dialog turn on the TReK Receipt Response (TRR) column. Push **OK** to close the dialog. Your main window should look similar to the one shown in Figure 3.



**Figure 3 Command Processing Main Window with TRR Column**

You should now see a new column called TRR in the command track area (it's the 4<sup>th</sup> column from the left). Since you're already familiar with command responses you can probably guess what this is all about. When you send a command to a TReK destination it will respond with a TReK Receipt Response. This will let you know that the TReK system (command node) received your command request and whether it was accepted or rejected. As you might guess you can use the View Realtime Commanding Messages dialog to see the information in the TRR.

2. Now it is time to send a command from the sub node and have the command node validate the contents of the command. Go to the **Command** menu and select **Commands**. In the **Commands** dialog select the **CAMERA\_MODE** command and push the **Modify** button. We will need to set a value for the **MODE** field before we can uplink the command. You should see the Modify Command dialog as shown in Figure 4.

Name: CAMERA\_MODE Header Name: HEADER01 Type: M Complete: No

Technical Name:

Description: Set camera mode.

Name	Type	Complete	Input Data Type	Uplink Data Type	Swap	Calibrator
CCSDS_CHECKSUM	P	Yes	S	ICLK	N	
EXP_DEST_FC	P	Yes	H	IUNS	N	
EXP_HDR_VERSION	P	Yes	H	IUNS	N	
EXP_SOURCE_FC	P	Yes	H	IUNS	N	
MODE	M	No	H	SUND	N	
MSG_BYTE_COUNT	P	Yes	H	IUNS	N	
MSG_TYPE	P	Yes	H	IUNS	N	

OK Modify Field... Cancel Help

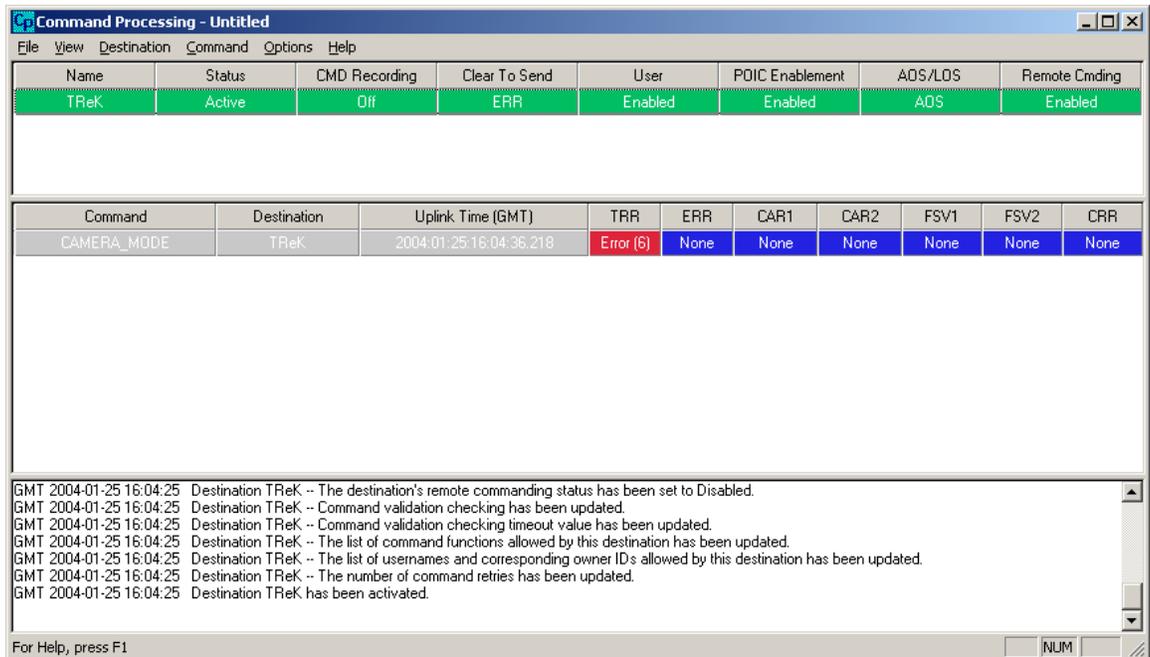
**Figure 4 Modify Command Dialog**

3. Select the **MODE** field in the list and push the **Modify Field** button. Change the field value to 0x2D as shown in Figure 5. You should enter the value without the “0x” prefix. TReK already knows that you are entering a hexadecimal value based on the selected Field Value Representation.

**Figure 5 Modify Command Field Dialog**

Press the **OK** button to change the value and return to the **Modify Command** dialog. Press the **OK** button on the **Modify Command** dialog to return to the **Commands** dialog. You are now ready to send the command.

- Now select the **CAMERA\_MODE** command in the **Commands** dialog and push the **Uplink Local Command** button. You should get a TRR error number 6 back from the command node in a couple of seconds as shown in Figure 6. TRR error number 6 indicates that a timeout error occurred while the command node was validating the command. This happened since we never started the user provided validation software. If you turn on user provided validation and your program crashes unexpectedly, no one will be able to send a command until you are able to restart your validation software.



**Figure 6 Command Track (Validation Timeout)**

- Return to the command node to start the user provided validation software. Go to the Windows Start menu of the Command Node. Select **Programs**, select **TReK** (this name may be different if you selected a different name during installation), select **Examples**, select **Visual C++**, and select **Command Validation** to start the user provided validation software.

Appendix A contains the portion of the validation code that does the checking of the command and returns the results to TReK. The validation software is set up to reject the command **CAMERA\_MODE** if it is sent by UserA and 46<sup>th</sup> byte of the data has a value of 0x2D. UserB is allowed to set the value of the 46<sup>th</sup> byte only to 0x33. All other users who have access to this command do not have restrictions on the value for any portion of the command.

The source code for this program can be found in the CommandValidation.cpp file located in the following folder:

<TReK Install Path>\Examples\Visual C++\CommandValidation

- Return to the sub node and resend the **CAMERA\_MODE** command. Select **CAMERA\_MODE** in the **Commands** dialog and press the **Uplink Local Command** button. This time a TRR message with error 5 is returned. TRR error 5 indicates that the command node validation failed. We set the value for the **MODE** field to 0x2D and the user provided validation software is set up to reject the command if the 46<sup>th</sup> byte is 0x2D for UserA.

7. The error number set by the user provided validation software is returned as part of the TRR message. You can see this value on the sub node by selecting the TReK destination in the main window and bringing up the viewer window. Go to the **Destination** menu and select **View Realtime Commanding Messages**. Scroll up in the dialog to find the last TRR message. It should be the second to last message in the viewer. Figure 7 shows the TRR message in the viewer window. The value returned by the user validation (26) can be found a few lines from the bottom of the dialog.

```

-----
TRT 2004-01-25 16:05:48.562
Hexadecimal Output
Size of data is 36
      00 01 02 03 04 05 06 07 08 09 0123456789
0000 18 00 C0 00 00 1D 40 13 E9 73 .....@..s
0001 00 26 C9 06 01 00 00 02 00 00 .&.....
0002 00 00 00 00 00 00 00 00 00 00 .....
0003 1A 00 05 00 EA D1 .....

POIC Command Message Headers
CCSDS Primary Header Values
Version:                0
Type:                   1
Secondary Header Flag:  1
Message Source:         Remote Command System (0)
Sequence Flags:         Unsegmented Data (3)
Sequence Count:         0
Message Length:         29

CCSDS Secondary Header Values
Coarse Time:            Wed Jan 29 16:06:11 2014
Fine Time:              0
Activity:               Other (0)
Checksum:               1
Unused bit:             0
Message Type:           Command Response (6)
Message Subtype:        TReK Receipt Response (TRR) (201)
Destination:            POIC (6)
User Defined Data:      16777218

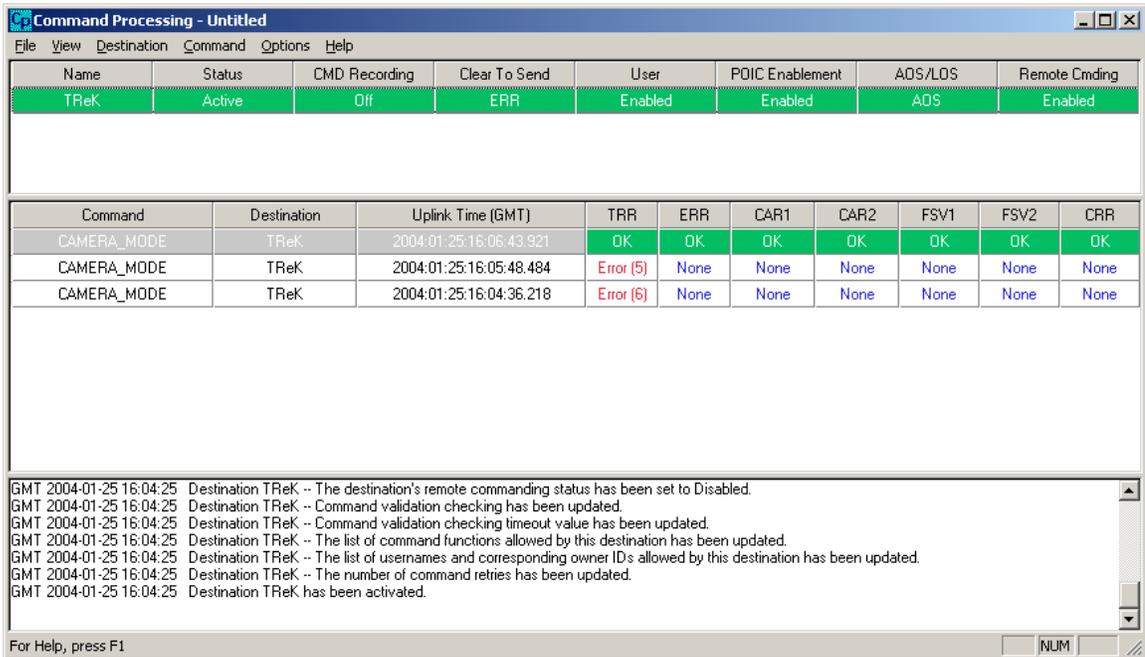
TRR Command Response Specific Data
Reflected Packet Sequence Count: 0
Coarse Time:            Sun Jan 06 00:00:00 1980
Fine Time:              0
Acknowledgment Code:    User Validation Error Found (5)
Validation Error:       26
Unused:                 0

Checksum for message
Checksum:               EA D1
-----
Close  Pause  Resume  Clear  Help

```

Figure 7 Realtime Commanding Messages (TRR Error 5)

8. Go back to the **Commands** dialog, select the **CAMARA\_MODE** command and push the **Modify** button. Change the value of the **MODE** field to 0x33. Don't forget that you shouldn't enter the "0x" prefix. See steps 2 and 3 from above if you don't remember how to change a field value in a command.
9. From the **Commands** dialog, select the **CAMERA\_MODE** command and push the **Uplink Local Command** button. The command should be accepted by the command node and forwarded to the destination. All of the responses received by the command node for this command will be routed back to the sub node. The main window on the sub node should now look like Figure 8.



**Figure 8 Command Track (Success!!!)**

## Appendix A Command Validation Source Code Snippet

```

//
//   Begin CommandValidation example code.
//

int return_value;
char *event_name_ptr = NULL;
HANDLE validation_event_handle;
BOOL loop = TRUE;
DWORD code;
char *username_ptr = NULL;
char *mnemonic_ptr = NULL;
unsigned char *cmd_data_ptr = NULL;
unsigned int cmd_data_length;
unsigned short validation_results;

// Get the event name from TReK for a destination named "POIC"
return_value = GetValidationEventName( "POIC", &event_name_ptr );

if( return_value != SUCCESS )
{
    printf( "Error getting validation event name.  Exiting program.\n" );
    return 1;
}

// Create/Open the event.  It is important to set the first three
// parameters to the CreateEvent WIN32 API call as shown below.
validation_event_handle = CreateEvent( NULL,
                                     NULL,
                                     NULL,
                                     event_name_ptr );

if( validation_event_handle == NULL )
{
    printf( "Error creating validation event.  Exiting program.\n" );
    return 1;
}

//
// Enter a loop for validation.  The loop will only exit if an error
// is encountered.
//

while( loop )
{
    // Wait for the validation event to occur.
    code = WaitForSingleObject( validation_event_handle, INFINITE );

    if( code != WAIT_OBJECT_0 )
    {
        printf( "Error occurred while waiting for signal.\n" );
        loop = FALSE;
    }
    else

```

```

{
    // Get the uplink packet from TReK.
    return_value = GetUplinkPacketForValidation( "POIC",
                                                &username_ptr,
                                                &mnemonic_ptr,
                                                &cmd_data_ptr,
                                                &cmd_data_length );

    if( return_value != SUCCESS )
    {
        printf( "Error getting uplink packet for validation." );
        loop = FALSE;
    }
    else
    {
        //
        // Validate the Command.
        //

        validation_results = 1; // Success if no errors are found

        if( !strcmp( mnemonic_ptr, "CAMERA_MODE" ) )
        {
            if( cmd_data_length != 48 )
                validation_results = 23; // command must be 48 bytes
            else
            {
                if( !strcmp( username_ptr, "UserA" ) )
                {
                    if( cmd_data_ptr[45] == 0x2D )
                        validation_results = 26; // UserA cannot use 0x26
                }
                else if( !strcmp( username_ptr, "UserB" ) )
                {
                    if( cmd_data_ptr[45] == 0x33 )
                        validation_results = 27; // UserB cannot use 0x33
                }
            }
        }

        // Send the results back to TReK
        return_value = SendValidationResults( "POIC",
                                            validation_results );

        if( return_value != SUCCESS )
        {
            printf( "Error returning validation results to TReK." );
            loop = FALSE;
        }
    }
}

//
// Clean up before exiting.
//

```

```
delete event_name_ptr;  
CloseHandle( validation_event_handle );
```

## Appendix B Glossary

Note: This Glossary is global to all TReK documentation. All entries listed may not be referenced within this document.

Application Programming Interface (API)	A set of functions used by an application program to provide access to a system's capabilities.
Application Process Identifier (APID)	An 11-bit field in the CCSDS primary packet header that identifies the source-destination pair for ISS packets. The type bit in the primary header tells you whether the APID is a payload or system source-destination.
Calibration	The transformation of a parameter to a desired physical unit or text state code.
Communications Outage Recorder	System that captures and stores payload science, health and status, and ancillary data during TDRSS zone of exclusion.
Consultative Committee for Space Data Systems (CCSDS) format	Data formatted in accordance with recommendations or standards of the CCSDS.
Consultative Committee for Space Data Systems (CCSDS) packet	A source packet comprised of a 6-octet CCSDS defined primary header followed by an optional secondary header and source data, which together may not exceed 65535 octets.
Conversion	Transformation of downlinked spacecraft data types to ground system platform data types.
Custom Data Packet	A packet containing a subset of parameters that can be selected by the user at the time of request.
Cyclic Display Update Mode	A continuous update of parameters for a particular display.
Decommutation (Decom)	Extraction of a parameter from telemetry.
Discrete Values	Telemetry values that have states (e.g., on or off).

Dump	During periods when communications with the spacecraft are unavailable, data is recorded onboard and played back during the next period when communications resume. This data, as it is being recorded onboard, is encoded with an onboard embedded time and is referred to as dump data.
Enhanced HOSC System (EHS)	Upgraded support capabilities of the HOSC systems to provide multi-functional support for multiple projects. It incorporates all systems required to perform data acquisition and distribution, telemetry processing, command services, database services, mission support services, and system monitor and control services.
Exception Monitoring	A background process capable of continuously monitoring selected parameters for Limit or Expected State violations. Violation notification is provided through a text message.
Expected State Sensing	Process of detecting a text state code generator in an off-nominal state.
EXPRESS	An EXPRESS Rack is a standardized payload rack system that transports, stores and supports experiments aboard the International Space Station. EXPRESS stands for EXpedite the PRocessing of Experiments to the Space Station.
File transfer protocol (ftp)	Protocol to deliver file-structured information from one host to another.
Flight ancillary data	A set of selected core system data and payload health and status data collected by the USOS Payload MDM, used by experimenters to interpret payload experiment results.

Grayed out	Refers to a menu item that has been made insensitive, which is visually shown by making the menu text gray rather than black. Items that are grayed out are not currently available.
Greenwich Mean Time (GMT)	The solar time for the meridian passing through Greenwich, England. It is used as a basis for calculating time throughout most of the world.
Ground ancillary data	A set of selected core system data and payload health and status data collected by the POIC, which is used by experimenters to interpret payload experiment results. Ground Ancillary Data can also contain computed parameters (pseudos).
Ground receipt time	Time of packet origination. The time from the IRIG-B time signal received.
Ground Support Equipment (GSE)	GSE refers to equipment that is brought in by the user (i.e. equipment that is not provided by the POIC).
Ground Support Equipment Packet	A CCSDS Packet that contains data extracted from any of the data processed by the Supporting Facility and the format of the packet is defined in the Supporting Facility's telemetry database.
Huntsville Operations Support Center (HOSC)	A facility located at the Marshall Space Flight Center (MSFC) that provides scientists and engineers the tools necessary for monitoring, commanding, and controlling various elements of space vehicle, payload, and science experiments. Support consists of real-time operations planning and analysis, inter- and intra-center ground operations coordination, facility and data system resource planning and scheduling, data systems monitor and control operations, and data flow coordination.

IMAQ ASCII	A packet type that was added to TReK to support a very specific application related to NASA's Return to Flight activities. It is not applicable to ISS. It is used to interface with an infrared camera that communicates via ASCII data.
Limit Sensing	Process of detecting caution and warning conditions for a parameter with a numerical value.
Line Outage Recorder Playback	A capability provided by White Sands Complex (WSC) to play back tapes generated at WSC during ground system communication outages.
Measurement Stimulus Identifier (MSID)	Equivalent to a parameter.
Monitoring	A parameter value is checked for sensing violations. A message is generated if the value is out of limits or out of an expected state.
Parameter	TReK uses the generic term parameter to mean any piece of data within a packet. Sometimes called a measurement or MSID in POIC terminology.
Payload Data Library (PDL)	An application that provides the interface for the user to specify which capabilities and requirements are needed to command and control his payload.
Payload Data Services Systems (PDSS)	The data distribution system for ISS. Able to route data based upon user to any of a number of destinations.
Payload Health and Status Data	Information originating at a payload that reveals the payload's operational condition, resource usage, and its safety/anomaly conditions that could result in damage to the payload, its environment or the crew.
Payload Operations Integration Center (POIC)	Manages the execution of on-orbit ISS payloads and payload support systems in coordination/unison with distributed International Partner Payload Control Centers, Telescience Support Centers (TSC's) and payload-unique remote facilities.

Payload Rack Checkout Unit (PRCU)	The Payload Rack Checkout Unit is used to verify payload to International Space Station interfaces for U.S. Payloads.
Playback	Data retrieved from some recording medium and transmitted to one or more users.
Pseudo Telemetry (pseudo data)	Values that are created from calculations instead of directly transported telemetry data. This pseudo data can be created from computations or scripts and can be displayed on the local PC.
Remotely Generated Command	A command sent by a remote user whose content is in a raw bit pattern format. The commands differ from predefined or modifiable commands in that the content is not stored in the POIC Project Command Database (PCDB).
Science data	Sensor or computational data generated by payloads for the purpose of conducting scientific experiments.
Subset	A collection of parameters from the total parameter set that is bounded as an integer number of octets but does not constitute the packet itself. A mini-packet.
Super sampled	A parameter is super sampled if it occurs more than once in a packet.
Swap Type	A flag in the Parameter Table of the TReK database that indicates if the specified datatype is byte swapped (B), word swapped (W), byte and word swapped (X), byte reversal (R), word reversal (V) or has no swapping (N).
Switching	A parameter's value can be used to switch between different calibration and sensing sets. There are two types of switching on TReK: range and state code.

Transmission Control Protocol (TCP)	TCP is a connection-oriented protocol that guarantees delivery of data.
Transmission Control Protocol (TCP) Client	A TCP Client initiates the TCP connection to connect to the other party.
Transmission Control Protocol (TCP) Server	A TCP Server waits for (and accepts connections from) the other party.
Telemetry	Transmission of data collected from a source in space to a ground support facility. Telemetry is downlink only.
Telescience Support Center (TSC)	A TSC is a NASA funded facility that provides the capability to plan and operate on-orbit facility class payloads and experiments, other payloads and experiments, and instruments.
User Application	Any end-user developed software program that uses the TReK Application Programming Interface software. Used synonymously with User Product.
User Data Summary Message (UDSM)	Packet type sent by PDSS that contains information on the number of packets sent during a given time frame for a PDSS Payload packet. For details on UDSM packets, see the POIC to Generic User IDD (SSP-50305).
Uplink format	The bit pattern of the command or file uplinked.
User Datagram Protocol (UDP)	UDP is a connection-less oriented protocol that does not guarantee delivery of data. In the TCP/IP protocol suite, the UDP provides the primary mechanism that application programs use to send datagrams to other application programs. In addition to the data sent, each UDP message contains both a destination port number and a fully qualified source and destination addresses making it possible for the UDP software on the destination to deliver the message to the correct recipient process and for the recipient process to send a reply.

User Product	Any end-user developed software program that uses the TReK Application Programming Interface software. Used synonymously with User Application.
Web	Term used to indicate access via HTTP protocol; also referred to as the World Wide Web (WWW).

## Appendix C Acronyms

Note: This acronym list is global to all TReK documentation. Some acronyms listed may not be referenced within this document.

AOS	Acquisition of Signal
API	Application Programming Interface
APID	Application Process Identifier
ASCII	American Standard Code for Information Interchange
CAR	Command Acceptance Response
CAR1	First Command Acceptance Response
CAR2	Second Command Acceptance Response
CCSDS	Consultative Committee for Space Data Systems
CDB	Command Database
CDP	Custom Data Packet
COR	Communication Outage Recorder
COTS	Commercial-off-the-shelf
CRR	Command Reaction Response
DSM	Data Storage Manager
EHS	Enhanced Huntsville Operations Support Center (HOSC)
ERIS	EHS Remote Interface System
ERR	EHS Receipt Response
EXPRESS	Expediting the Process of Experiments to the Space Station
ES	Expected State
FAQ	Frequently Asked Question
FDP	Functionally Distributed Processor
FSV	Flight System Verifier
FSV1	First Flight System Verifier
FSV2	Second Flight System Verifier
FPD	Flight Projects Directorate
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
GRT	Ground Receipt Time
GSE	Ground Support Equipment
HOSC	Huntsville Operations Support Center
ICD	Interface Control Document
IMAQ ASCII	Image Acquisition ASCII
IP	Internet Protocol
ISS	International Space Station
LDP	Logical Data Path
LES	Limit/Expected State
LOR	Line Outage Recorder
LOS	Loss of Signal
MCC-H	Mission Control Center – Houston
MOP	Mission, Operational Support Mode, and Project
MSFC	Marshall Space Flight Center
MSID	Measurement Stimulus Identifier

NASA	National Aeronautics and Space Administration
OCDB	Operational Command Database
OS	Operating System
PC	Personal Computer, also Polynomial Coefficient
PCDB	POIC Project Command Database
PDL	Payload Data Library
PDSS	Payload Data Services System
PGUIDD	POIC to Generic User Interface Definition Document
POIC	Payload Operations Integration Center
PP	Point Pair
PRCU	Payload Rack Checkout Unit
PSIV	Payload Software Integration and Verification
RPSM	Retrieval Processing Summary Message
SC	State Code
SCS	Suitcase Simulator
SSP	Space Station Program
SSCC	Space Station Control Center
SSPF	Space Station Processing Facility
TCP	Transmission Control Protocol
TReK	Telescience Resource Kit
TRR	TReK Receipt Response
TSC	Telescience Support Center
UDP	User Datagram Protocol
UDSM	User Data Summary Message
URL	Uniform Resource Locator
USOS	United States On-Orbit Segment
VCDU	Virtual Channel Data Unit
VCR	Video Cassette Recorder
VPN	Virtual Private Network