

# OASIS-CC

*Operations and Science Instrument Support  
Command and Control*

## **Quick Reference Manual**

For OASIS-CC V02.05.12  
and subsequent versions  
November 1994

# OASIS-CC

*Operations and Science Instrument Support  
Command and Control*

## Quick Reference Manual

Permission is granted to make and distribute verbatim copies of this manual provided that the copyright notice and these paragraphs are preserved on all copies. The software and any accompanying written materials are provided “as is” without warranty of any kind. The University of Colorado disclaims all warranties of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose.

Mention of any commercial company or product in this document does not constitute an endorsement by the University of Colorado.

# Table of Contents

Introduction.....	1-1
Creating Telemetry Items.....	2-1
Items Representing Numeric (Analog) Values .....	2-1
Example Raw Integer Requiring Calibration to EU.....	2-1
Example Raw Integer Requiring Multi-Segmented Calibration to EU.....	2-1
Example Raw Integer - no Calibration Desired.....	2-2
Example Pre-Calibrated Real.....	2-2
Items Representing STATES.....	2-2
Example Raw Integers Converted to STATES.....	2-2
Example STATES Converted to Raw Integers.....	2-3
Items reflecting STRING data.....	2-3
Example Character Strings.....	2-3
Defining LIMIT Checking.....	3-1
Setting Primary HILO Thresholds.....	3-1
Example HILO Limits.....	3-1
Setting Secondary HILO Thresholds.....	3-2
Example Secondary HILO Limits.....	3-2
Setting Delta Thresholds.....	3-2
Example Delta Limits.....	3-2
Setting Trend Time Limits.....	3-2
Example Trend Limits.....	3-3
Defining STATE Checking.....	4-1
Setting Desirability.....	4-1
Example Setting Desirability.....	4-1
Defining TRIGGERS.....	5-1
Example Creating a TRIGGER .....	5-1
Defining EQUATIONS.....	6-1
Example Creating an Equation .....	6-1
Defining Commands.....	7-1
Designing the CSTOL Command Statements .....	7-1
Command Verbs and Objects.....	7-1
Attributes .....	7-2

Subfield Clauses..... 7-2  
Example Creating a CSTOL Command Statement..... 7-3  
Defining a Hierarchy of Commandable Elements..... 7-4  
Example Creating a Command Hierarchy..... 7-4  
Defining Command Channels..... 7-6  
Example Creating a Command Channels ..... 7-6  
Defining the Command Data..... 7-7  
Example Binary Commands..... 7-8  
Example ASCII Commands.....7-10  
Creating a Wildcard Display ..... 8-1  
Example Creating a Wildcard Page..... 8-1  
Performance Considerations.....A-1

## List of Figures

Figure 1. Panel Titled wildcard viewed from the AWB.....	8-2
Figure 2. Panel Titled insert.....	8-3
Figure 3. Panel Titled delete.....	8-3
Figure 4. CSTOL Procedure wildcard.prc .....	8-5

## List of Tables

Table 1 Binary and ASCII Command Tables.....	7-8
Table 2. Item Attributes on the Wildcard Panel.....	8-1
Table 3. Actions for Wildcard Inputs.....	8-4
Table 4. Database Items Required for the wildcard Panel.....	8-5

This page is intentionally blank.

# 1. Introduction

The OASIS-CC Quick Reference Manual provides quick access to information application managers frequently require. It also presents features not covered in other OASIS-CC documents. If the information you seek is not present, consult the other OASIS-CC documents. Consider sending your suggestions for additional topics to include.

Many examples are presented, showing how to fill the database tables to achieve desired functions. In all cases, database information is presented in the form of CSTOL query *insert* and *update* directives.

This document is one of a suite that includes:

*OASIS-CC CSTOL Reference Manual: LASP*

*OASIS-CC Database Guide: LASP*

*OASIS-CC System Manager's Guide: LASP*

*OASIS-CC Installation Guide: LASP*

*OASIS-CC Application Environment Reference Manual: LASP*

*OASIS-CC Generic Communications User's Guide: LASP*

*Actions Workbench User's Guide: LASP*

*TAE+ Overview: Century Computing, Inc., Laurel, MD*

*TAE+ User Interface Developer's Guide: Century Computing, Inc., Laurel, MD.*

This page is intentionally blank.

## 2. Creating Telemetry Items

### Items Representing Numeric (Analog) Values

#### **Example      Raw Integer Requiring Calibration to EU**

Consider a thermocouple reading from a spectrometer's detector assembly. The data is telemetered as a binary integer that is converted to an engineering unit.

```
insert LATEST_DATA &
    external_element=DETECTOR, item_name=TEMPERATURE, &
    data_class=NUMERIC,units=C
insert ANALOG_CONVERSIONS &
    external_element=DETECTOR, item_name=TEMPERATURE, &
    eu_units=C, segment=1, is_last=TRUE, c0="0.0", c1="2.34", c2="-.34"
```

When the first segment is the last, the coefficients apply to the entire range of raw values.

#### **Example      Raw Integer Requiring Multi-Segmented Calibration to EU**

Consider a transducer that measures a pressure. The calibration of the item requires a segmented conversion because one set of coefficients cannot adequately represent the conversion from the integer to EU value.

```
insert LATEST_DATA external_element=INTERNAL, item_name=PRESSURE,&
    data_class=NUMERIC,units=T
insert ANALOG_CONVERSIONS &
    external_element=INTERNAL, item_name=PRESSURE, eu_units=T, &
    dn_low=1,segment=40, is_last=FALSE, c0="0.0", c1="2.34", c2="-.34"
insert ANALOG_CONVERSIONS &
    external_element=INTERNAL, item_name=PRESSURE, eu_units=T, &
    dn_low=85,segment=2, is_last=FALSE, c0="0.0", c1="1.23", c2="-.1"
```

```
insert ANALOG_CONVERSIONS &
    external_element=INTERNAL, item_name=PRESSURE, eu_units=T, &
    dn_low=255,segment=3, is_last=TRUE
```

In multi-segmented conversions, DN\_LOW defines the upper bound of applicable integer values to which the previous coefficient set applies. The coefficients in the last record are ignored.

### **Example      Raw Integer - no Calibration Desired**

Consider a grating drive stepper motor reading. It arrives as a binary integer that requires no calibration. This is denoted by choosing units of DN (Data Number).

```
insert LATEST_DATA &
    external_element=GRATING, item_name=POSITION, &
    data_class=NUMERIC,units=DN
```

When the units are specified as DN (Data Number), the ANALOG\_CONVERSIONS table is not used, however the raw value is automatically converted to a float, which is placed in the EU field.

### **Example      Pre-Calibrated Real**

Consider a current measurement that is telemetered as a floating point number that is already calibrated to represent amperes.

```
insert LATEST_DATA &
    external_element=PRIMARY_BUS, item_name=CURRENT, &
    data_class=NUMERIC,units=A
```

An ANALOG\_CONVERSION record is unnecessary since the value is already calibrated.

## **Items Representing STATES**

### **Example      Raw Integers Converted to STATES**

Consider a discrete measurement that represents the status of a detector's high voltage supply. A telemetered zero represents off, a one represents on. If the value is two, assume that for some reason, the status is unavailable.

```
insert LATEST_DATA &
    external_element=HIGH_VOLTAGE, item_name=STATUS, &
    data_class=STATE
```

```
insert STATE_CONVERSIONS,&
  external_element=HIGH_VOLTAGE, item_name=STATUS,&
  dn_low=0, dn_high=0, state=OFF, desirability=GOOD

insert STATE_CONVERSIONS,&
  external_element=HIGH_VOLTAGE, item_name=STATUS,&
  dn_low=1, dn_high=0, state=ON, desirability=GOOD

insert STATE_CONVERSIONS,&
  external_element=HIGH_VOLTAGE, item_name=STATUS,&
  dn_low=2, dn_high=0, state=UNKNOWN, desirability=GOOD
```

DN\_HIGH must be supplied in an insert since it is a field which makes up the record's key, however the field is not currently used by OASIS-CC.

### **Example STATES Converted to Raw Integers**

Items that arrive as formatted character strings can be processed as states, as long as their length within their data stream is constant. The definition is identical to Example 1. A DN\_LOW must be assigned to each state that will be received.

## **Items reflecting STRING data**

### **Example Character Strings**

Consider a measurement that consists of ASCII character data. OASIS-CC stores 80 bytes of character data.

```
insert LATEST_DATA &
  external_element=TEXT, item_name=MESSAGE, &
  data_class=CHARACTER_STRING
```

This page is intentionally blank.

## 3. Defining LIMIT Checking

LIMIT checking provides high/low and delta comparisons for numeric items. High/low (HILO) checking compares a telemetry point's current value to pre-established thresholds. Delta checking determines how much a current value changed from its last value, and compares the difference to a pre-established limit.

### Setting Primary HILO Thresholds

#### Example HILO Limits

Consider the thermocouple reading defined in the previous chapter, DETECTOR TEMPERATURE. Assume its anticipated normal range is -15.0 (C) to +20.0 (C). As long as the temperature remains within this range, the detector is at a safe temperature. Also assume that if the temperature falls below -20.0 (C) or above +30.0 (C), the detector may be damaged. OASIS-CC is given this information with the following statements:

```
insert LIMITS &
  external_element=DETECTOR, item_name=TEMPERATURE, &
  units=C, red_low="-20.0", red_high="30.0", yellow_low="-15.0", &
  yellow_high="20.0"
update LATEST_DATA &
  hilo_enabled=TRUE where external_element=DETECTOR and &
  item_name=TEMPERATURE
```

The update directive used to enable the HILO checking is required because the insert directive used to create the LATEST\_DATA item omitted this field (thus relying on the default value *FALSE*.)

If DETECTOR TEMPERATURE stays within the yellow thresholds, an alpha-numeric display of the item will appear green. Otherwise, it will appear yellow or red, depending on which threshold is exceeded. At any threshold transition, a message is issued to the message queue indicating such (the message is colored according to the threshold crossed).

## Setting Secondary HILO Thresholds

Operations may require that subsequent alarms are issued when an item remains within a yellow or red alarm regime (the thresholds defined so far issue messages only when a threshold is crossed). OASIS-CC provides relative thresholds for this purpose.

### Example      Secondary HILO Limits

Assume that operators are to be alerted if the DETECTOR TEMPERATURE changes by more than 1 Celsius degree once within a yellow low or yellow high range. Similarly, assume alerts are to be issued if an already red DETECTOR TEMPERATURE changes by more than 0.5 since the last red alert. Add this information to the existing record in the following manner:

```
update LIMITS &
  msg_delta_yellow="1.0", msg_delta_red="0.5" where &
  external_element=DETECTOR and item_name=TEMPERATURE
```

## Setting Delta Thresholds

Certain telemetry measurements convey significant information not only in their absolute value, but also in how much the value changed since its last update.

### Example      Delta Limits

Consider the PRIMARY\_BUS CURRENT monitor described earlier. Assume that if this measurement changes by more than 0.8 ampere since it's last update, an alert must be issued. Provide OASIS-CC with this information, and modify the existing LATEST\_DATA record to enable delta limit checking:

```
insert LIMITS &
  external_element=PRIMARY_BUS, item_name=CURRENT, &
  units=A, delta_limit="0.8"
update LATEST_DATA &
  delta_enabled=TRUE where external_element=PRIMARY_BUS and &
  item_name=CURRENT
```

Delta alerts are always red, but they never initiate triggers. See Defining a Trigger.

## Setting Trend Time Limits

OASIS-CC provides the ability to calculate the rate of change of an item and alert the user if the item appears headed for a red high or red low crossing within a specified time interval.

### **Example      Trend Limits**

If operators want to know that the DETECTOR TERMPATURE will go red high 5 minutes in advance, inform OASIS-CC:

```
update LIMITS &
  trend_time_limit="300.0", points_in_trend=64 where &
  external_element=DETECTOR and item_name=TEMPERATURE
update LATEST_DATA &
  trend_enabled=TRUE where external_element=DETECTOR and &
  item_name=TEMPERATURE
```

Note that points\_in\_trend must be sufficiently large to provide a smooth extrapolation, yet not too large to unnecessarily consume memory. A red warning message is issued if the projected red limit crossing will occur in less than the specified trend\_time\_limit.

This page is intentionally blank.

## 4. Defining STATE Checking

STATE checking reports if a discrete item has changed its value, or the desirability of the value has changed, or both.

### Setting Desirability

All of the integer values that a state item can take must have a translation to a literal value like ON or OFF or OPEN or CLOSED. Each translation must have a desirability of either GOOD or BAD.

#### Example      Setting Desirability

The HIGH\_VOLTAGE STATUS was defined previously in Items that Represent a STATE, and each translation was given a GOOD desirability. Perhaps it was decided after the fact that when the high voltage was enabled, operators required the associated alphanumeric display to appear red. The following modification provides for this change:

```
update STATE_CONVERSIONS,&
    desirability=BAD where external_element=HIGH_VOLTAGE &
    and item_name=STATUS and state=ON
update LATEST_DATA &
    state_check_class=BOTH where &
    external_element=HIGH_VOLTAGE, item_name=STATUS &
    data_class=STATE
```

Setting state\_check\_class=BOTH enables OASIS-CC to issue a message of the appropriate color (red or green) whenever HIGH\_VOLTAGE status changes its value or desirability. If state\_check\_class is set to DESIRABILITY, no message will be generated if the HIGH\_VOLTAGE STATUS changes from OFF to UNKNOWN, since these values share the same desirability.

This page is intentionally blank.

## 5. Defining TRIGGERS

Triggers are automated reactions to the crossing of a red HILO limits threshold, or the change from a GOOD to a BAD state. Triggers are defined for the appropriate items in LATEST\_DATA (except those that have a data\_class of character\_string), and the reactions are defined as a CSTOL statement, usually *START PROCNAME*. Multiple LATEST\_DATA records can share the same trigger.

### Example      Creating a TRIGGER

Assume that if the DETECTOR TEMPERATURE exceeds +30.0 (C), commands to disable the detector heaters are to be automatically transmitted - no questions asked.. (See Items that Represent a Numeric Value, and Defining LIMITS Checking). Assume the CSTOL commands TURN OFF AFT HEATER and TURN OFF FORE HEATER accomplish this action. Provide OASIS-CC with this information:

```
update LATEST_DATA &
  trigger_name=SAFE_HEATER_TEMP &
  where external_element=DETECTOR and item_name=TEMPERATURE

insert TRIGGERS name=SAFE_HEATER_TEMP, enabled=TRUE,&
  statement="START SAFE_HEATER"

insert procedures procedure_name=SAFE_HEATER,filename="safe_heater.prc"
```

The body of \$OASIS\_PROCS/safe\_heater.prc might be:

```
proc safe_heater
begin

; the following determines if the trigger was activated on a red high... this
; logic must be consistent with the limits definitions...

if detector temperature > 20.0 C
  turn off aft heater
  turn off fore heater
else
  ;do nothing
endif
```

```
update triggers enabled=TRUE where name=SAFE_HEATER_TEMP  
endproc
```

Note the last action of `safe_heater.prc` is to re-enable the trigger. OASIS-CC automatically disables a trigger once it has executed to prevent noisy data from initiating it repetitively. In this example, the safeguard is explicitly overruled by the trigger procedure itself.

When the trigger executes the first time, the procedure will compile and execute on a control language processor reserved for triggers – the trigger CLP. Once a procedure is compiled on the trigger CLP, it can never be decompiled (during the OASIS-CC session). Consider testing the source from the User CLP before enabling the trigger.

## 6. Defining EQUATIONS

Equations are the mechanism used to automatically evaluate pseudo-telemetry points. Pseudo-telemetry points are measurements that don't come directly from data streams, but reflect measurements based on telemetered data.

Defining an equation is similar to defining a trigger, however the mechanism that initiates an equation is different. Equations are initiated when a specific global variable, or LATEST\_DATA record, changes its value. The equation is named by the EQUATION\_NAME field in the LATEST\_DATA record used to drive the equation. Whenever the item's value changes (which may be at a lower frequency than when a new value is received), the equation ultimately references a CSTOL statement to be executed. Usually, the CSTOL statement consists of a call to start a CSTOL procedure, since most equations require more than one line of CSTOL to complete.

### Example      Creating an Equation

Consider two telemetered current measurements, HEATER\_1 CURRENT and HEATER\_2 CURRENT. Operations require a view of the total current these devices draw, but it's not available in the telemetry. Assume LATEST\_DATA records exist for both currents (standard numeric items with UNITS=A), however no equation has been implemented.

Instruct OASIS-CC to initiate an equation whenever either available current changes:

```
update LATEST_DATA equation_name=TOTAL_CURRENT where &  
external_element=HEATER_1 and item_name=CURRENT
```

```
update LATEST_DATA equation_name=TOTAL_CURRENT where &  
external_element=HEATER_2 and item_name=CURRENT
```

Note that both items reference the same equation. Create a new LATEST\_DATA record to store the value of the total current:

```
insert LATEST_DATA external_element=TOTAL, item_name=CURRENT, &  
data_class=NUMERIC,units=A
```

Provide a reference to a CSTOL statement to be initiated by the TOTAL\_CURRENT equation:

```
insert EQUATIONS name=TOTAL_CURRENT, &
    enabled = FALSE, statement="start add_currents"
```

The equations will be enabled later. (Note that equations can also be implemented in either C or Ada. This is discussed in Chapter 4 of the *OASIS-CC System Manager's Guide*. For the purposes of this example, the equation is implemented in CSTOL.) Since OASIS-CC doesn't have a record of the procedure *total\_currents*, supply it:

```
insert PROCEDURES procedure_name=ADD_CURRENTS, &
    filename="add_currents.prc"
```

All that's left is to implement the procedure code. A listing of the procedure \$OASIS\_PROCS/add\_currents.prc might look like this:

```
proc add_currents

;Declare a real variable to work with. In general it's best to use
;local variables to perform the math
declare variable $tmp_total = 0.0
declare variable $tmp_icurrent1 = 0.0
declare variable $tmp_icurrent2 = 0.0

;fill the local variables with the global variables
let $tmp_icurrent1 = 1./(HEATER_1 CURRENT)
let $tmp_icurrent2 = 1./(HEATER_2 CURRENT)
let $tmp_total = $tmp_icurrent1+$tmp_icurrent2

;invert back to a current
let $tmp_total=1./$tmp_total

;and place the value back into LATEST_DATA
let TOTAL CURRENT = $tmp_total

endproc
```

Prior to enabling the equation, test it by starting it from the User CLP:

```
start total_currents
```

If it's buggy, edit the source, decompile and start the procedure until it has the desired affect.

```
decompile total_currents
start total_currents
```

When it works, enable the equation. When an equation is started by changing the value of a LATEST\_DATA item that names it, the procedure will compile and execute on a control language processor reserved for equations – the equation CLP. Once a procedure is

compiled on the equation CLP, it can never be decompiled (during the OASIS-CC session).

update equations enabled=TRUE where name=TOTAL\_CURRENTS

This page is intentionally blank.

## 7. Defining Commands

Implementing command definition in the database involves:

- 1) Designing the CSTOL command statements. When entered into a control language processor (CLP), these statements initiate the transmission of the appropriate bit pattern. A pre-defined command statement syntax must be followed.
- 2) Defining a hierarchy of commandable elements. OASIS-CC allows multiple command channels - or communications interfaces, and all commands sent on a given channel make up a branch of the hierarchy. The names of the commandable elements are defined by the CSTOL statements designed in (1).
- 3) Defining command channel(s) by specifying the communication devices, protocols, etc.
- 4) Defining the command data associated with the CSTOL command statements.

OASIS-CC can generate command data in binary or ASCII format. Steps (1) through (3) are identical when implementing either type of command. Step (4) requires different steps for binary and ASCII commands.

### Designing the CSTOL Command Statements

#### Command Verbs and Objects

CSTOL requires a certain syntax for all command statements. The full command directive construct will make itself known as this discussion progresses. To start, the most simple representation of the command syntax is:

*Verb Object like close relay\_1, or set grating\_position.*

The verbs CSTOL provides are listed in Chapter 9 of the *OASIS-CC CSTOL Reference Manual*. If desired, the application manager can expand this list. Verbs translate into the NAME of the command as it is known by the command database tables. Objects can be anything you want, limited to 16 characters, and translate into the EXTERNAL\_ELEMENT of the command as known by the command tables.

## Attributes

To improve the readability of command directives, and to reduce the proliferation of underscores, the command syntax provides an optional Attribute to help describe the Object. Now the syntax appears as:

*Verb Object [Attribute]*

like *close relay one* or *set grating position*. Attributes are optional when the command directive is designed. If used, they must always be supplied when the command directive is issued. If an attribute is used, the command tables' NAME becomes a concatenation of the Verb, an underscore, and the attribute – all of which cannot exceed 16 characters. The command tables' NAME would be *set\_position*, and the EXTERNAL\_ELEMENT, *grating*.

## Subfield Clauses

Some commands are discrete: their bit pattern never alters. Others contain subfields that can take on varying data. OASIS-CC command directives allow subfields to be filled with the TO, BY, FROM and WITH clauses. Discrete commands do not use these clauses, nor the database information that must back them up. The command directive syntax is completely specified below. (Actually, there are further twists to the syntax that are not discussed in this document. Consult Chapter 10 of the *OASIS-CC System Managers Guide* for the complete syntax specification).

*Verb Object [Attribute] [TO x]*

*Verb Object [Attribute] [BY x]*

*Verb Object [Attribute] [FROM x]*

*Verb Object [Attribute] [WITH A w [,B x [,C y...and Z z]]]*

TO, BY, and FROM are known to the command database tables as SUBFIELD\_NAMES. A, B, C, ...Z also translate as SUBFIELD\_NAMES, however the command designer chooses their spelling. The spelling of all italicized words in the command syntax is up to the designer. (Verbs are chosen from the list referenced above). The subfield delineators (,) and (and) are interchangeable in the WITH clause.

The user supplies the subfield values w,x,y, and z when the command is issued. These values can be an integers, or EUs ( Engineering Units – real numbers with units), or a STATES. Example commands directives with subfields:

set grating position to 745

set grating position to lyman\_alpha

set transponder with channel q, mode a and ranging off

The TO, BY and FROM allow users to pass only one subfield value when the command is issued. The WITH clause does not limit the number of subfields that can be supplied when the command is issued.

After CSTOL command directives are implemented in the database, they are fixed. That is, none of the elements of the command statement are interchangeable. (The subfield values are not fixed, and all or parts of the subfield clauses can be omitted if the supplied default subfield values are acceptable. The order in which subfield values are supplied does not matter.)

### **Example      Creating a CSTOL Command Statement**

Consider the command pattern:

FAF320A329900012324A (hex)

This command turns on a heater (as governed by the value 99), and sets a thermostat according to the values shown currently as zeros. Another representation of the command:

FAF320A320100012324A (hex)

turns off the heater. This command has two subfields: one controls whether the heater is on or off, the other controls how hot it should get when on. The thermostat control data is ignored by the receiver if the command turns off the heater. After considerable review (and emotion), the control review team designed a CSTOL statement that will issue the above bit patterns. The winning selection was:

*set heater with mode on and temp xxx*  
and  
*set heater with mode off*

According the CSTOL command syntax, the commandable element is the *heater*; no attribute was used, so the command name is *set*. Subfield names are *mode* and *temp*. The subfield value for *mode* can be either 'on' or 'off', and is supplied when the statement is entered by the user. The subfield value for *temp* is an integer, and is supplied when the command is entered.

The runner up statement design, and just as doable is:

*set h with m n, t xxx*  
and  
*set h with m f*

where the commandable element is *h*, command name is *set*, the subfield names are *m* (with allowable values 'n' or 'f'), and *t* (expecting an integer value following).

## Defining a Hierarchy of Commandable Elements

All commandable elements must be represented in the database within a hierarchy. The table that stores the hierarchy is ELEMENT\_HIERARCHIES. The hierarchy is used to

- 1) Provide a map of commandable elements that a given user class may access. User classes are defined in the USER\_PRIVILEGES table, and have an associated element name. A user class has access to the named element, and all elements below it in the hierarchy.
- 2) Define command channel elements. Command channels are associated with a specific communications device. Commands sent on a given channel will have commandable elements in the hierarchy that branch below a command channel element. Note that for bus-type of communications devices (such as IEEE\_488), each device on a bus acts as a command channel element. For communication devices not on a bus (such as serial ports) one command channel element exists that reflects the interface.
- 3) Define a topmost element that provides access to all elements in the hierarchy for the MASTER user class.

### Example Creating a Command Hierarchy

Consider the selected CSTOL command statements and their bit patterns from the example in Designing the CSTOL Command Statements:

*set heater with mode on and temp xxx*                    (FAF320A329900012324A hex)

and

*set heater with mode off*                                    (FAF320A320100012324A hex)

Invent a name to be used as the command channel element in ELEMENT\_HIERARCHIES. Command channel elements are often named by the object that will receive the commands. Typically, this device is a command decoder of sorts: pick 'SCCDU', for *spacecraft command decoder unit*. So far, The hierarchy looks like this:

SCCDU

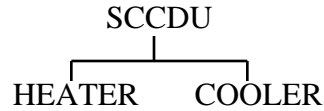
Since the heater is serviced by SCCDU, the hierarchy becomes:

```

SCCDU
 |
HEATER

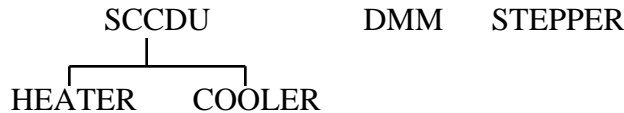
```

To further demonstrate the hierarchy, assume that SCCDU also services a cooler:

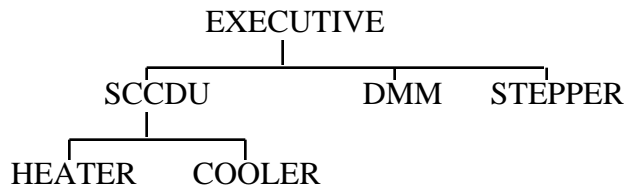


Most OASIS-CC applications have multiple command channels, so pretend there is another device to which an entirely different flavor of commands are sent. This device is an IEEE bus that services a slew of test equipment, like a digital multimeter and a stepper motor. The IEEE equipment needs to be defined in the table slightly differently – each device on the bus acts as a command channel element (it needs to be its own oldest ancestor). OASIS-CC knows which bus – there can be more than one – the DMM and STEPPER are on by information provided in the ELEMENT\_CHARACTERISTICS table.

The hierarchy now looks like:



Since command privileges for a class of user are associated with a named element and all of the elements below it, this hierarchy won't allow any user to command both the digital multimeter, the stepper, and the heater from the same OASIS-CC session. (Only one user class can run OASIS-CC at a given time). To rectify this, a last element is needed, and it's role is to provide full command access:



Fill the ELEMENT\_HIERARCHIES table, starting with the top most element (although the order in which the inserts are executed doesn't matter):

```

insert ELEMENT_HIERARCHIES name=EXECUTIVE, &
    oldest_ancestor=EXECUTIVE, parent='OASIS$$SYSTEM'
  
```

EXECUTIVE is arbitrarily named. The top most element requires that name=oldest\_ancestor and parent=OASIS\$\$SYSTEM. Next, create the command channel elements. The bit patterns for commandable elements that are offspring of a command channel element leave OASIS-CC via the same interface using the same protocol, etc. Command channel elements are identified by name = oldest\_ancestor.

```

insert ELEMENT_HIERARCHIES name=SCCDU, &
    oldest_ancestor=SCCDU, parent =EXECUTIVE
insert ELEMENT_HIERARCHIES name=DMM, &
    oldest_ancestor=IDMM, parent =EXECUTIVE
insert ELEMENT_HIERARCHIES name=STEPPER, &
    oldest_ancestor=STEPPER, parent =EXECUTIVE

```

Finally, define the commandable elements. Their command channel element is defined by the oldest ancestor. Since this example only implements one tier below the command channel elements, the parents and oldest\_ancestors happen to be the same.

```

insert ELEMENT_HIERARCHIES name=HEATER, &
    oldest_ancestor=SCCDU, parent =SCCDU
insert ELEMENT_HIERARCHIES name=COOLER, &
    oldest_ancestor=SCCDU, parent =SCCDU

```

Recall that *HEATER* reflects the object of the CSTOL command statements *set heater with...* All objects in CSTOL command statements

## Defining Command Channels

In the previous section, command channel elements are identified in the ELEMENT\_HIERARCHIES table. All command channel elements require a characteristics definition, a forward stream definition, and a command message definition.

### Example Creating a Command Channels

Consider the CSTOL command statements, their associated bit patterns, and the command channel element from the previous Section's examples:

```

    set heater with mode on and temp xxx      (FAF320A329900012324A hex)
and
    set heater with mode off                  (FAF320A320100012324A hex)
and
    Command channel name                      (SCCDU)

```

All command channels require one record in ELEMENT\_CHARACTERISTICS that among other things, associates a forward stream name for the command channel. Define this association, and define the forward stream:

```

insert ELEMENT_CHARACTERISTICS name=SCCDU, &
    stream_name=SDCDU_OUTPUT
and

```

```
insert STREAMS stream_name=SDCDU_OUTPUT,&  
processor=EXTCOMM_SUBSYSTEM,direction=FORWARD_STREAM,&  
stream_type=PRIMARY,link_name=SDCDU_OUT_LINK,&  
protocol=IP
```

Depending on the protocol employed by the forward stream, certain parameters in the STREAMS and ELEMENT\_CHARACTERISTICS may or may not apply. In this example, the IP (Internet) protocol was specified. The full set of information required for this protocol is defined in the *OASIS-CC Generic Communications User's Guide*. Specialized protocols (like IEEE) also require specific database information. Consult the appropriate release notes for the database information the protocol requires.

Every command channel requires a record in COMMAND\_MESSAGES to define common header patterns, trailer patterns, min and max lengths, etc. In the example, both commands share a similar headers and trailers. Assume all commands that are transmitted on this channel share these attributes:

```
insert COMMAND_MESSAGES external_element=SCCDU,&  
message_type=REALTIME, max_length=80, header_length=24, &  
header_pattern="FAF320", trailer_length=24, trailer_pattern="12324A"
```

## Defining the Command Data

OASIS-CC allows for the construction of either binary or ASCII commands. In either case, all commands leaving OASIS-CC are strings of bits. The primary difference between binary and ASCII commands resides in how the command information is defined in the database, and how user-supplied subfield values are translated into the command data prior to transmission. Whenever ASCII commands are mentioned, it is assumed that the majority of the command data has a meaningful character interpretation. Otherwise, the command is known as a binary.

The approach to implementing a command differs for the two types. For example, Table 1 shows that the two types don't use the exact same database table sets. Binary commands are defined by first describing the command as a fixed length bit pattern, and second, mapping into the fixed pattern any subfields the command requires (if any). ASCII commands are defined by concatenating a series of character segments, some of which may or may not be subfields. The process for defining both types follows.

**Table 1 Binary and ASCII Command Tables**

<b>Binary Commands use (at least):</b>	<b>ASCII Commands use (at least):</b>
COMMANDS and, depending on the command's complexity: COMMAND_MAPS  COMMAND_VALUES COMMAND_VALUE_CONVERSIONS COMMAND_STATE_CONVERSIONS	COMMANDS ASCII_COMMAND_MAPS  and depending on the command's complexity: ASCII_COMMAND_VALUES COMMAND_VALUE_CONVERSIONS COMMAND_STATE_CONVERSIONS

**Example Binary Commands**

Consider the command information presented earlier:

FAF320A329900012324A (hex)

This command turns on a heater (as governed by the value 99), and sets a thermostat according to the values shown currently as zeros. Another representation of the command:

FAF320A320100012324A (hex)

turns off the heater. This command has two subfields: one controls whether the heater is on or off (99 vs. 01), the other controls how hot it should get when on. The thermostat control data is ignored by the receiver if the command turns off the heater.

Since the commands' headers and trailers are already identified in COMMAND\_MESSAGES (see the previous section), the data yet to define is:

*set heater with mode on and temp xxx* (A3299000 hex)

and

*set heater with mode off* (A3201000 hex)

All commands (binary or ASCII) require a COMMANDS record:

```
insert COMMANDS external_element=HEATER,&
name=SET, is_legal_in_realtime_msg=TRUE, legal_source=CLP,&
cmd_type=IMMEDIATE, safety_level=SAFE, is_discrete=FALSE, &
is_ascii=FALSE, length=32, fixed_pattern="A3201000"
```

HEATER is the commandable element in ELEMENT\_HIERARCHIES. For a binary command (is\_ascii=FALSE), is\_discrete=FALSE implies subfields are used, and

therefore, a record(s) in COMMAND\_VALUES and COMMAND\_MAPS. If this command were discrete, the database definition would be complete.

Define the characteristics of the subfields *mode* and *temp*:

```
insert COMMAND_VALUES external_element=HEATER,&
  command_name=SET, subfield_name=MODE, &
  max_value="153.0", min_value="1.0",default_value="1.0",&
  default_safety_level=SAFE, mapping_format=INTEGER
```

```
insert COMMAND_VALUES external_element=HEATER,&
  command_name=SET, subfield_name=TEMP,&
  max_value="4095.0", min_value="0.0",default_value="0.0",&
  default_safety_level=SAFE,mapping_format=INTEGER
```

The max\_values 153.0 and 4095.0 correspond to integer 99 (hex) and 12 bits of ones. (Note that the value is mapped as INTEGER data. Had the value needed to be mapped in single or double precision format, the MAPPING\_FORMAT would be IEEE\_FLOAT – 32 bits, or IEEE\_LONG\_FLOAT – 64 bits). If any of the command subfields are omitted in the CSTOL statement, the default values are used. The example makes the following assumptions: the heater turn on must be explicit, and; if not explicitly turned on (MODE ON), the command sets the bits to turn off the heater, in which case the receiver ignores the thermostat setting anyway. The resulting command patterns when combinations of subfields are omitted are::

<i>set heater with mode on and temp 4095</i>	(A3299FFF hex)	heater on, full temp
<i>set heater with mode off</i>	(A3201000 hex)	heater off
<i>set heater with temp 4095</i>	(A3201FFF hex)	heater off
<i>set heater</i>	(A3201000 hex)	heater off

OASIS-CC needs to know where to place the subfields in the commands' fixed pattern:

```
insert COMMAND_MAPS external_element=HEATER,&
  command_name=SET, subfield_name=MODE, cmd_type=IMMEDIATE,&
  segment=1, is_last=TRUE, source_first_bit=1, source_last_bit=8,&
  destination_first_bit=13, destination_last_bit=20
```

```
insert COMMAND_MAPS external_element=HEATER,&
  command_name=SET, subfield_name=TEMP, cmd_type=IMMEDIATE,&
  segment=1, is_last=TRUE, source_first_bit=1, source_last_bit=12,&
  destination_first_bit=21, destination_last_bit=32
```

The source bits refer to positions in the subfield value, the destination bits refer to positions in the fixed pattern. Subfield values overwrite the bits defined in the fixed pattern. If desired, the bits supplied in the subfield value can be split and placed in non-contiguous locations of the fixed pattern. This requires one record in COMMAND\_MAPS for each segment of the divided subfield value. The length of the value is determined by the sum of

the differences (one difference per segment) of the source and destination bit coordinates. INTEGER values can be 1 to 32 bits long, whereas single precision and double precision values must be 32 and 64 bits long.

Since the legal values for the MODE subfield are *on* or *off* a translation is required to turn the literals into values:

```
insert COMMAND_STATE_CONVERSIONS external_element=HEATER,&
      commmand_name=SET, subfield_name=MODE, state=ON,&
      value=153
```

```
insert COMMAND_STATE_CONVERSIONS external_element=HEATER,&
      commmand_name=SET, subfield_name=MODE, state=OFF,&
      value=1
```

Although not demonstrated, the COMMAND\_VALUE\_CONVERSIONS is similar to the COMMAND\_STATE\_CONVERSIONS table. If desired and applicable, EUs (Engineering Units – real numbers with a units specifier) can be supplied as subfield values in a CSTOL command statement. If EUs are supplied, there needs to be a record(s) in COMMAND\_VALUE\_CONVERSIONS used to perform a single or segmented polynomial conversion to translate the real into an integer value that's inserted into the command. A good example is setting a grating drive motor to a position sensitive to a popular wavelength:

*set grating position to 1024.5 A (Angstroms)*

which could be equivalent to:

*set grating position to 123*

Currently, OASIS-CC can only format command data as ASCII data or binary integer data.

### **Example      ASCII Commands**

Whereas binary commands are fixed patterns into which subfields are mapped, ASCII commands are series of character segments concatenated to form the command. The individual segments defined can be fixed (the value never changes), or subfields (the value is supplied by the CSTOL command statement).

Consider a command intended for the COOLER (see the example in Defining a Hierarchy of Commandable Elements). In order to demonstrate an ASCII command, assume that the cooler expects the following command data:

```
FAF320 "C1<CR>" 12324A
```

and

```
FAF320 "C2<CR>" 12324A
```

These commands set the cooler's operations to mode 1 and mode 2. As established in the previous sections, all commands traveling the same channel share header and trailer information, which is stored in the COMMAND\_MESSAGES table. Further, the existing COMMAND\_MESSAGES record shows that no SCCDU command can be longer than 80 bits. The cooler commands are each 9 bytes long, including the header and trailer.

The control review team decided on the following CSTOL command statement to issue the above data:

```
set cooler mode to m
```

In this CSTOL command statement, the commandable element is *cooler*, *mode* is an attribute that modifies the command name to *set\_mode*. The subfield name is *to*, and *m* is either "1" or "2" (ASCII).

All commands (binary or ASCII) require a COMMANDS record:

```
insert COMMANDS external_element=COOLER,&
      name=SET_MODE, is_legal_in_realtime_msg=TRUE, legal_source=CLP,&
      cmd_type=IMMEDIATE, safety_level=SAFE, is_discrete=FALSE, &
      is_ascii=TRUE
```

The length of the complete command will be based in the length of the constituent segments. ASCII commands require that `is_discrete=FALSE`.

Characterize the subfield value that can be either "1" or "2":

```
insert ASCII_COMMAND_VALUES external_element=COOLER,&
      command_name=SET_MODE, subfield_name="TO",&
      format="I1", default_substring_value="2.0", default_substring_length=1,&
      default_safety_level=HAZARDOUS, max_value="2.0", min_value="1.0"
```

The `leading_...` and `trailing_...` fields (not shown) allow control over how to format the command output if the supplied value consumes fewer characters than will be allocated by the format specification. This subfield will always consume 1 character. If the subfield is omitted in the CSTOL command statement, the instrument will be set to mode 2, and the operator will have to answer an alert requesting clearance for relying on the default value (`default_safety_level=HAZARDOUS`).

Assemble the segments into the command string:

```
insert ASCII_COMMAND_MAPS external_element=COOLER,&
      command_name=SET_MODE, cmd_type=IMMEDIATE,&
      segment=1, is_last=FALSE, is_a_subfield=FALSE, is_ascii=TRUE,&
      substring_value="C", substring_length=1
```

Is\_a\_subfield=FALSE because the value is not supplied by the CSTOL command statement – it's supplied in this database record. Is\_ascii=TRUE says how the value for this non-subfield segment will be input - ASCII or hex.

```
insert ASCII_COMMAND_MAPS external_element=COOLER,&  
command_name=SET_MODE,cmd_type=IMMEDIATE,&  
segment=2, is_last=FALSE, is_a_subfield=TRUE, subfield_name=TO
```

Since this segment is a subfield, the length and data information are calculated based on the value input by the CSTOL command statement, and parameters set for this subfield in ASCII\_COMMAND\_VALUES.

```
insert ASCII_COMMAND_MAPS external_element=COOLER,&  
command_name=SET_MODE,cmd_type=IMMEDIATE,&  
segment=3, is_last=TRUE, is_a_subfield=FALSE, is_ascii=FALSE,&  
binary_substring_value="0D", substring_length=1
```

Carriage returns <CR> cannot be entered as ASCII characters in the substring\_value, so its hex equivalent is supplied in binary\_substring\_value.

## 8. Creating a Wildcard Display

In applications where LATEST\_DATA has many records, it's useful to have the ability to display items that are not included on any pre-designed panels. To implement this capability, create a wildcard page that allows the user to specify LATEST\_DATA items during run-time.

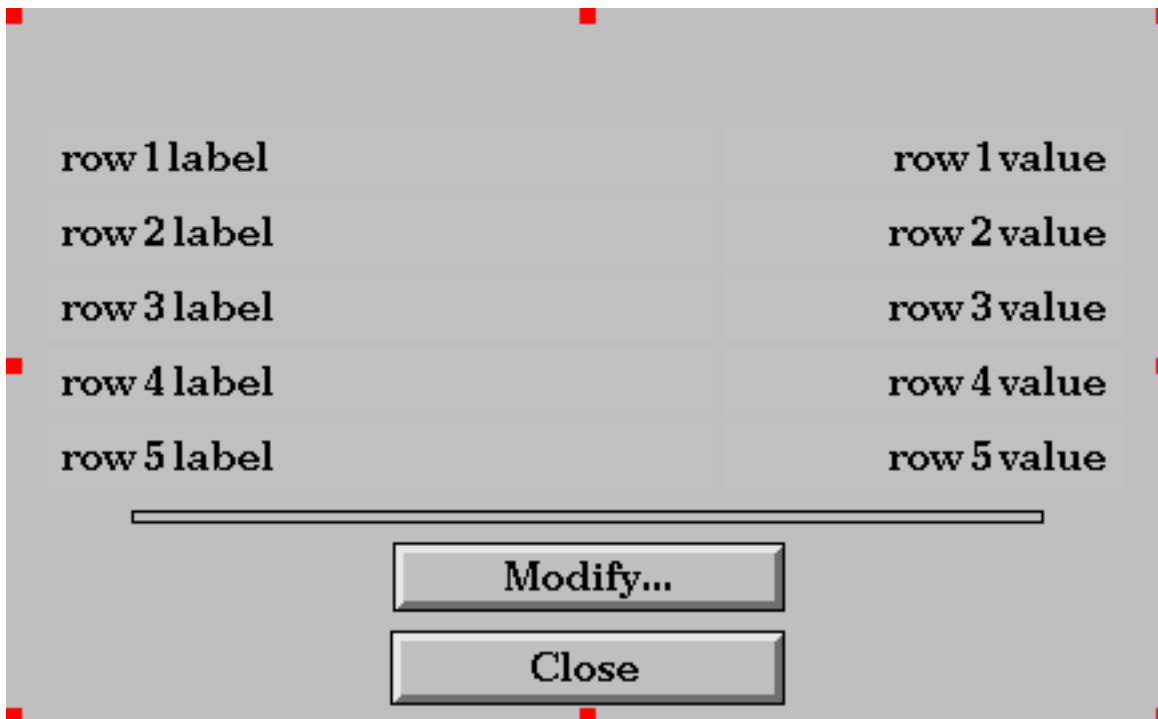
### Example Creating a Wildcard Page

From the Actions Workbench, create a panel (call it *wildcard*). This panel will display two LATEST\_DATA items for each wildcard element: a label, and a data value from the desired telemetry point. On the panel, create an even number of *dynamic text* items, and give them the attributes listed in Table 2.

**Table 2. Item Attributes on the Wildcard Panel**

TAE Item Name	Default String	Data Type
wlabel1	row 1 label	string
wlabel2	row 2 label	string
wlabel3	row 3 label	string
wlabel4	row 4 label	string
wlabel5	row 5 label	string
wcv1	row 1 value	string
wcv2	row 2 value	string
wcv3	row 3 value	string
wcv4	row 4 value	string
wcv5	row 5 value	string

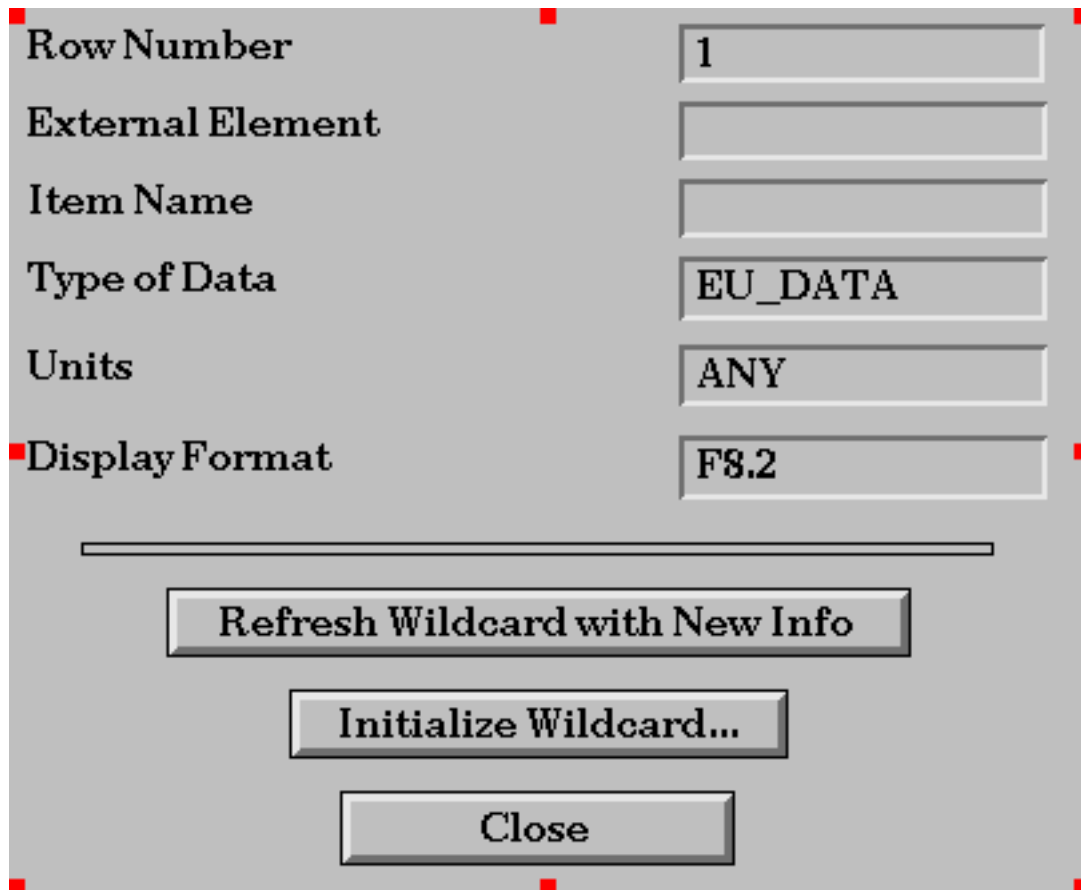
Arrange the items on the wildcard panel as shown in Figure 1:

**Figure 1. Panel Titled *wildcard* viewed from the AWB**

Define a connection for the *Modify...* push button to bring up the panel shown in Figure 2. Create another panel (name is *wc* with a title *insert*) as shown in Figure 2. This panel will be used to name the desired telemetry points you want to display, and assemble a label to go along with it, and refresh the actual wildcard display. Define a connection to clear the wildcard panel with the button *Close*.

When developing the panel *wc*, name the text keyin item used to define the desired row *m*; the keyin for external\_element *ee*; Item\_name in; Type of Data *dt*; Units *vu*; and Value Format *vf*.

Define a connection for the *Initialize Wildcard...* button to raise another panel that requests approval for this action. This Panel is shown in Figure 3

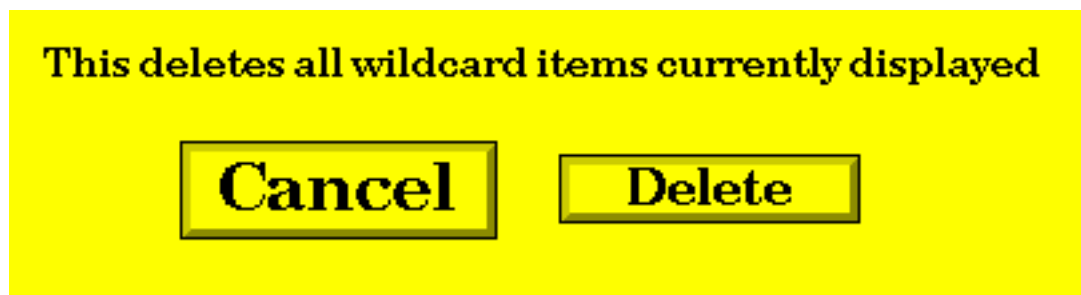
Figure 2. wc Panel Titled *insert*

The screenshot shows a dialog box with a light gray background. It contains the following fields and buttons:

- Row Number:** A text box containing the value "1".
- External Element:** An empty text box.
- Item Name:** An empty text box.
- Type of Data:** A text box containing the value "EU\_DATA".
- Units:** A text box containing the value "ANY".
- Display Format:** A text box containing the value "F8.2".

Below the input fields, there are three buttons arranged vertically:

- Refresh Wildcard with New Info:** A wide button with a dark border.
- Initialize Wildcard...:** A button with a dark border.
- Close:** A button with a dark border.

Figure 3. wc Panel Titled *delete*

Place the AWB in Define Connections mode, and provide the Actions presented in Table 3 for the buttons titled *Refresh Wildcard with New Info* and *delete*.

**Table 3. Actions for Wildcard Inputs**

<b>Tae Item Title</b>	<b>Actions (don't include the comments).</b>
Refresh Wildcard with New Info	<pre>-- Wpt_NewPanel goes here Cstolq "let wc_display label_ "+wc.rn+"=" "+wc.ee+" "+wc.in+""" Cstolq "delete display_definitions where tae_base=wcv"+wc.rn Cstolq "insert display_definitions (note: continued on next line _ this should occupy only one line)       external_element="+wc.ee+",item_name="+wc.in+",type_of_data="+wc.dt+",index=0,tae_base=wcv"+wc.rn Cstolq "update display_definitions value_format="+wc.vf+",units="+wc.vu+" where tae_base=wcv"+wc.rn Cstolq "clear wildcard" Cstolq "display wildcard"</pre>
Delete	<pre>-- Wpt_NewPanel goes here Cstolq "clear wildcard" Cstolq "init_wildcard" Cstolq "display wildcard"</pre>

Note that the second CSTOL statement sent to the system by the *Delete* push button is a CSTOL Macro named *init\_wildard*. The body of this macro is explained below. The Cancel button should clear the *delete* panel. Generate the interface code, recompile, and relink (these steps are required whenever a new panel is added, or any connections or actions within the interface have been modified).

Create and run a CSTOL procedure that creates the following the database entries shown in Table 4. The body of such a procedure, *create\_wildcard.prc* is shown in Figure 4.

**Table 4. Database Items Required for the *wildcard* Panel**

<b>LATEST_DATA</b> Slots for character strings that will be used as labels	<b>DISPLAY_DEFINITIONS</b> To display the strings held in the LATEST_DATA records to the left.	<b>DISPLAY_DEFINITIONS</b> To display to-be-named LATEST_DATA items
external_element =wc_display item_name =label_1 data_class =character_string	tae_base =wclabel1 external_element =wc_display item_name =label_1 type_of_data =eu_data	tae_base =wcvalue1 external_element =wc_display item_name =dummy type_of_data =eu_data
external_element=wc_display item_name = label_2 data_class = character_string	tae_base =wclabel2 external_element =wc_display item_name =label_2 type_of_data =eu_data	tae_base =wcvalue1 external_element =wc_display item_name =dummy type_of_data =eu_data
external_element=wc_display item_name =label_3 data_class =character_string	tae_base =wclabel3 external_element =wc_display item_name =label_3 type_of_data =eu_data	tae_base =wcvalue1 external_element =wc_display item_name =dummy type_of_data =eu_data
external_element=wc_display item_name =label_4 data_class =character_string	tae_base =wclabel4 external_element =wc_display item_name =label_4 type_of_data =eu_data	tae_base =wcvalue1 external_element =wc_display item_name =dummy type_of_data =eu_data
external_element=wc_display item_name =label_5 data_class =character_string	tae_base =wclabel5 external_element =wc_display item_name =label_5 type_of_data =eu_data	tae_base =wcvalue1 external_element =wc_display item_name =dummy type_of_data =eu_data
external_element=wc_display item_name =dummy data_class =character_string		

**Figure 4. CSTOL Procedure create\_wildcard.prc**

```

proc create_wildcard

:INSERT THE MACRO THAT DUPLICATES THIS PROCEDURE. THIS MACRO IS CALLED FROM THE
:INTERFACE TO REINITIALIZE THE WILDCARD DISPLAY
delete macros where macro_name=init_wildcard
insert macros macro_name=init_wildcard,filename="init_wildcard.mac"

:CREATE THE GLOBAL VARIABLES TO STORE LABELS FOR EACH ROW
delete latest_data where external_element=wc_display
insert latest_data external_element=wc_display,item_name=label_1,&
    
```

```

        data_class=character_string
insert latest_data external_element=wc_display,item_name=label_2,&
        data_class=character_string
insert latest_data external_element=wc_display,item_name=label_3,&
        data_class=character_string
insert latest_data external_element=wc_display,item_name=label_4,&
        data_class=character_string
insert latest_data external_element=wc_display,item_name=label_5,&
        data_class=character_string

;CREATE A STRING ITEM TO DISPLAY FOR UNUSED ROW VALUES
insert latest_data external_element=wc_display,item_name=dummy,&
        data_class=character_string

;INITIALIZE UNUSED LABELS AND VALUES WITH DEFAULT DATA
let wc_display label_1 = "row 1"
let wc_display label_2 = "row 2"
let wc_display label_3 = "row 3"
let wc_display label_4 = "row 4"
let wc_display label_5 = "row 5"
let wc_display dummy = "unused"

;CREATE DISPLAY_DEFINITIONS FOR THE (UNUSED) LABELS AND VALUE
delete display_definitions where external_element=wc_display
insert display_definitions tae_base=wclabel1, external_element=wc_display,&
        item_name=label_1,type_of_data=eu_data,index=0,value_format="a33",&
        show_units=false,show_data_type=false
insert display_definitions tae_base=wclabel2, external_element=wc_display,&
        item_name=label_2,type_of_data=eu_data,index=0,value_format="a33",&
        show_units=false,show_data_type=false
insert display_definitions tae_base=wclabel3, external_element=wc_display,&
        item_name=label_3,type_of_data=eu_data,index=0,value_format="a33",&
        show_units=false,show_data_type=false
insert display_definitions tae_base=wclabel4, external_element=wc_display,&
        item_name=label_4,type_of_data=eu_data,index=0,value_format="a33",&
        show_units=false,show_data_type=false
insert display_definitions tae_base=wclabel5, external_element=wc_display,&
        item_name=label_5,type_of_data=eu_data,index=0,value_format="a33",&
        show_units=false,show_data_type=false

delete display_definitions where tae_base=wc_v1
insert display_definitions tae_base=wc_v1, external_element=wc_display,&
        item_name=dummy,type_of_data=eu_data,index=0,value_format="a16"
delete display_definitions where tae_base=wc_v2
insert display_definitions tae_base=wc_v2, external_element=wc_display,&
        item_name=dummy,type_of_data=eu_data,index=0,value_format="a16"
delete display_definitions where tae_base=wc_v3
insert display_definitions tae_base=wc_v3, external_element=wc_display,&
        item_name=dummy,type_of_data=eu_data,index=0,value_format="a16"
delete display_definitions where tae_base=wc_v4
insert display_definitions tae_base=wc_v4, external_element=wc_display,&

```

```
        item_name=dummy,type_of_data=eu_data,index=0,value_format="a16"  
delete display_definitions where tae_base=wc5  
insert display_definitions tae_base=wc5, external_element=wc_display,&  
        item_name=dummy,type_of_data=eu_data,index=0,value_format="a16"  
  
write " "  
write "<G> The wildcard display (wildcard) is available..."  
write " "  
endproc
```

Make provisions for starting the `create_wildcard` procedure in your application. Also create a CSTOL Macro named `init_wildcard.mac`. The macro will be identical to the procedure `create_wildcard`, with the exception of the `macro init_wildcard` and `end macro` statements at the beginning and end of the source code. Once the procedure has executed, the user can:

- display both new panels.
- name a desired LATEST\_DATA record in keyins titled External\_Element and Item\_Name
- push *Refresh...* to enter the data into the database and redisplay the wildcard panel.
- push *Initialize Wildcard...* to clear any previous data entered.

This page is intentionally blank.

# Appendix A. Performance Considerations

To minimize unnecessary processing, consider these pointers:

Avoid using display update rates of 0.0. For most scenarios, a 1.0 second update rate is sufficient.

Combine equations whenever possible. Overhead is reduced by minimizing the number of procedures that execute.

Reduce the frequency at which equations execute. In the Defining Equations chapter, the example shows an equation calculate a total current when either of the constituent current measurements change value. If either constituent updates frequently, a lot of processing will occur to keep the calculated current as fresh. Is it necessary? Consider naming the equation with a telemetry item that updates at a lower frequency.

Avoid supplying values to protected fields when using the CSTOL insert directive. The results may be unpredictable.

If your application uses TAE Text Label items to display alphanumeric data, change them to Text Dynamic\_text. These items impose significantly less on the CPU, and have much smoother refresh characteristics.

If your application employs large CSTOL procedures, compile them just after startup. Compiling procedures takes a fair amount of work. Decompile procedures once they are used.

This page is intentionally blank.