PCA Instrument Commanding

This document illustrates design of the PCA Command Generation Software, which is a subsystem of the Science Operations Center (SOC) command generation subsystem for the X-Ray Timing Explorer (XTE).

PCA-CMD-DES-V4.5

XTE-PCA Software Module

November 5, 1995

by

Vikram Savkoor

Hughes STX Corp.

4400 Forbes Blvd., Lanham, MD 20706

1.0 Introduction

The PCA Command Subsystem allows a user to format a desired instrument configuration through an interactive window environment. The configuration is translated into a series of commands that are formatted as packets. The configuration is named and saved to disk for subsequent scheduling.

Section 2 briefly describes the requirements in the design of this subsystem. Section 3 lists the deliveries that will be made over time describing what parts of the design are to be delivered at different times. Section 4 describes the operating scenario. Section 5 discusses some open issues. Section 6 briefly describes some design features. Section 7 describes the interface between this subsystem and SOC's subsystems. Section 8 describes the subsystem design.

2.0 Requirements

Design of this subsystem is based on the description in SOC-102 and SOF Requirements document. These requirements are listed in the document entitled "Command Generation, Public Interfaces" by Leon Herreid. Since the PCA instrument team has agreed to this interface as specified by the SOC development team, the requirements need not be repeated here. This document also serves as an Interface Control Document (ICD) until such time as one is formally released.

3.0 Build Deliveries

Since this document is iterative over time, it is appropriate to put in a section on Build Deliveries that describes what parts of the design are to be delivered at different times. Therefore, this document reflects the overall design however not all of the completed design is to be delivered at once.

Applicable to all build deliveries are the following: (1) A User's Guide that describes normal operation of the software. (2) An Installation Guide that describes the installation and build procedures. (3) A Test Plan describing the procedures to follow for testing the software. (4) A Design Document that describes in detail the architecture of the software component. (5) Any applicable release notes. Depending on the size of the subsystem, these may be contained in one document or be presented as separate documents. If these are contained in one document, they will be listed out in separate sections. Since these are considered separate physical configuration items, it is mandatory that they accompany each build delivery. Delivery is not complete until they have been signed off by a member of the configuration management team.

3.1 Build 1

The Build 1 delivery is scheduled for July, 1993. The software for that delivery is due one month before or at the end of June, 1993. Included in this delivery is:

- working prototype of the Configuration Editor. This includes the user interface and software to generate a Command Script which is defined as a list of CCSDS formatted packets.
- Makefiles that automate the formation of the Build 1 executable.
- All five deliverables described above.

3.2 Build 2

The Build 2 delivery is scheduled for November, 1993. The software is due at the end of October, 1993. Included in this delivery is:

- A fully functional Discriminator, Test Pulse Generator, Configuration Summary, and Master Reset options.
- A fully functional Save, Load, and View features.
- A Browse Only Mode feature.
- Limited depth On line help.
- A Test Plan describing the procedures to follow while testing the software.
- An updated Design document.
- An updated User's Guide.

3.3 Build 3

The Build 3 delivery is scheduled for May, 1994. The software is due at the end of April, 1994. For this build all functionalities (including the above items) will be operational:

- The RIF Diagnostics and Constrained (HV) commands.
- Command Packet builder support for all commands.
- Constraints and Warnings.
- Configuration Editor output linked to PCA Housekeeping Telemetry Simulator.
- A Test Plan describing the procedures to follow while testing the software.
- An updated Design document.
- An updated User's Guide.

3.4 Build 3.1

The Build 3.1 delivery is scheduled for June, 1994. For this build all functionalities (including the above items) will be operational:

- Improved Constraints and Warnings System features.
- On-Line Help feature.
- PCA Windows ICON.
- A Test Plan describing the procedures to follow while testing the software.
- An updated Design document.
- An updated User's Guide.

3.5 Build 4

The Build 4 delivery is scheduled for November, 1994. It will include:

- Fully developed On-Line Help.
- Enhanced Editor code.
- Enhanced Command Packet builder code.
- A Test plan describing the procedures to follow while testing the software.
- An updated Design document.
- An updated User's guide.

4.0 Operating Scenario

The PCA instrument is commanded through the Remote Interface unit (RIF) by way of the Spacecraft Computer. Commands are sent in blocks destined for a particular PCU based on an unique identifier. These commands blocks are formatted on the ground and their contents are defined indirectly by members of the science team. Command blocks are scheduled according to guest observer requirements and uplinked when a command window is available. Each instrument team is provided an interface within the SOC environment to develop command blocks in an automated, consistent fashion.

The PCA instrument commanding is based on the notion of configurations. A configuration is the specified status of the instrument electronics at an anticipated point in time. Changes to the configuration can be translated into a series of commands that will program the instrument to a specific state. Instrument commands are grouped logically by function. There are four functional command sets for the PCA: (1) Discriminator Settings (2) Test Pulse Generator (TPG) Settings (3) Remote Interface Diagnostics and (4) Constrained Settings. The discriminator settings affect PHA measurements, the TPG commands affect the TPG operations, the RIF diagnostics relate to the testing or manual operation of the rate counter channels and hazardous settings usually deal with the high voltage or other commands that could damage the electronics if incorrect sequences of commands are repeatedly used.

It is envisioned that the duty scientist will be able to create new configurations through a windows-based interface. The interface will allow the scientist to change configuration settings based on knowledge of the instrument status at a specified time. Configuration settings will be changed through available menu selections thus reducing the chance of error by having to type them in individually. Once the settings have been completed, the software will generate a command list that will put the instrument in the desired configuration. The command list will contain commands formatted according to the CCSDS packet standard. The duty scientist gives the configuration a name and it is made available for scheduling using the mission scheduling software available within the SOC.

Warnings and constraint messages will be programmed into the software to warn against potentially hazardous or meaningless instrument conditions. These are considered to be internal to this section of code until time information is added to the configuration. Also, it is assumed that some sort of configuration naming convention will be adopted and reflected within this software component so that time ordering can be verified.

All hazardous commands will be processed with special confirmations so as to minimize the risk of possible error.

5.0 Open issues

- How is uplink command packet length determined? (Build 1)
- How is the checksum for the packet calculated? (Build 1)
- How will testing of this interface be performed? (All Builds)
- Screen layouts and GUI standards? (Build 1)
- What is the mechanism by which the duty scientist can read the measured and predicted configurations? (Build 2)
- How will warnings/constraints be handled? (Build 3)
- How are the constrained commands handled as opposed to other commands? (Build 2)
- How are configurations named and stored? Are there naming conventions that should be used? Are they stored in the database? If not, are they simple disk files and where do they reside? (Build 1)

6.0 Design Features

When a configuration is created, it is necessary to have both the measured (the last verified state of the instrument) and the predicted (what the configuration will be at a given point in time) configurations available for reference. Although the exact mechanism of this interface is still TBD, it is assumed that a tool will exist outside of the commanding software to readily display this information with the best accuracy possible. There are limitations in this in that the predicted configuration will never be guaranteed but can be used as a best-guess as to the instrument status. It is assumed that we will be able to import these values into the command interface.

7.0 Interfaces

The following indicate interfaces with other subsystems, Instrument Teams and external organizations.

7.1 Other SOC Subsystems

Since the PCA command software is instrument specific and therefore unique to the PCA, the command generation software described here will interact mainly with the Command Generation software provided in the SOC. It will also indirectly provide input to the Mission Scheduling software by creating named configuration objects. The Health and Safety Subsystem will also be accessed for measured Configuration values.

Since the ability to access the measured and predicted configuration information is still TBD, it is unclear what other SOC components will be involved.

8.0 Subsystem Design

The PCA Command Generation Software consists of three functional areas: (1) A Configuration Editor (2) A Configuration Builder and (3) a Command Builder. The Configuration Editor provides a windows-based interface to the user in order to create a new or edit an existing configuration. It consists of a series of screens from which various options for instrument settings are chosen. It relies on the Configuration Builder to physically create a new Desired Configuration Object or bring up and existing Predicted Configuration Object. The Configuration Builder also has the function of saving the configuration.

Once a configuration has been determined, the Command Builder is responsible for creating a Command Script. The Command Script is a series of commands that will put the instrument in the desired configuration. They are ordered and are formatted according to the CCSDS packet standard. No timing information is processed by the PCA Command Generation Software. Timing is added through a separate subsystem interface within the SOC.

The configuration editor consists of six screens, each serving a different functional purpose: (1) Command Selection (2) Discriminator Settings (3) Test Pulse Generator (TPG) Settings (4) Remote Interface Diagnostics (5) Constrained Command Settings (6) Configuration Summary (7) Master Reset Option. Screen layouts for some screens are depicted below. The editor as well as the command builder will issue warnings or constraints. Corrective action may be necessary in these cases.

Figure 3. shows an example screen layout for the main menu. All menus will be developed using TAE+: a NASA/GSFC developed software tool for building and generating code for user interfaces.

FIGURE 1. PCA Instrument Commanding: Object Classes

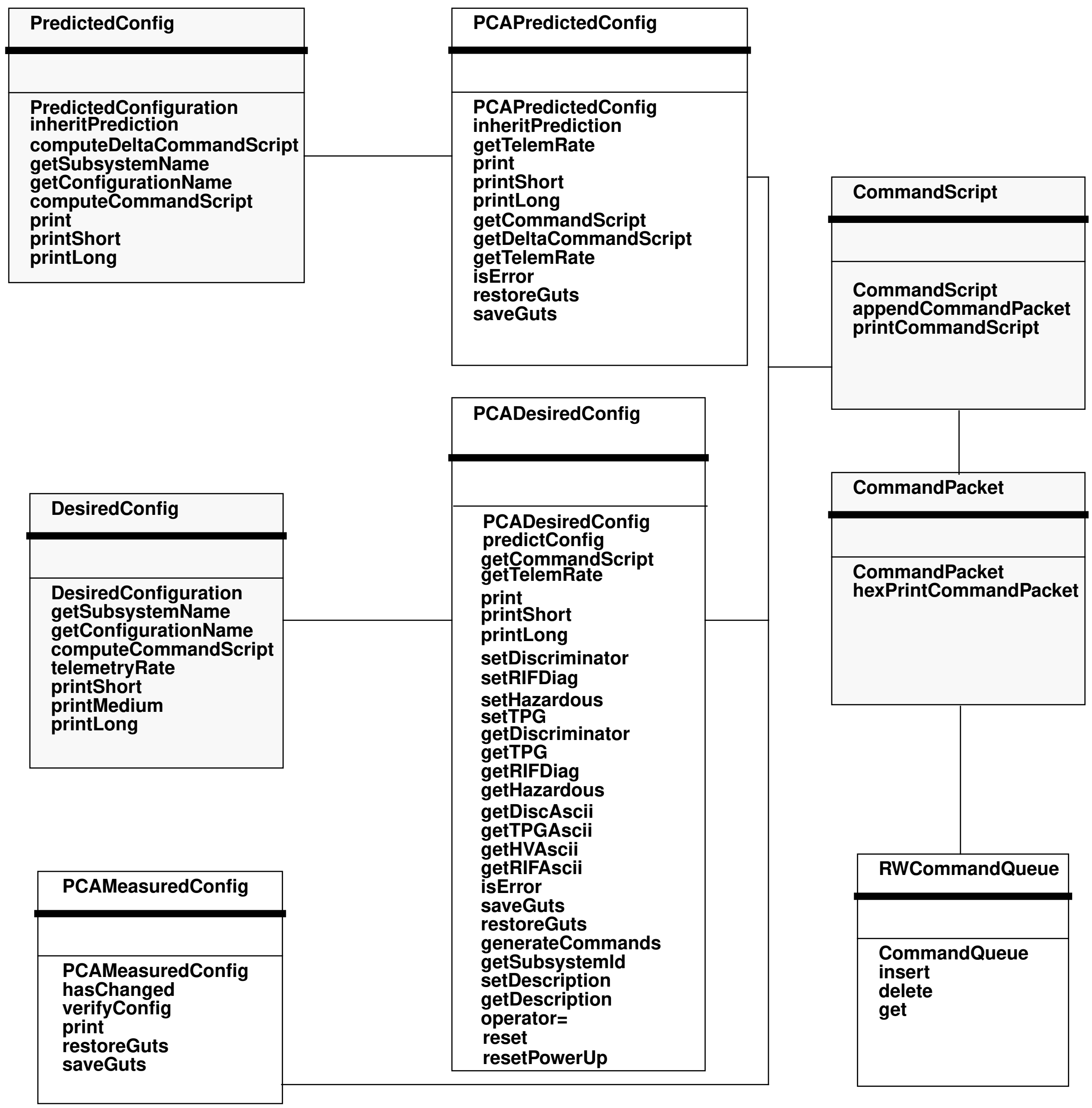
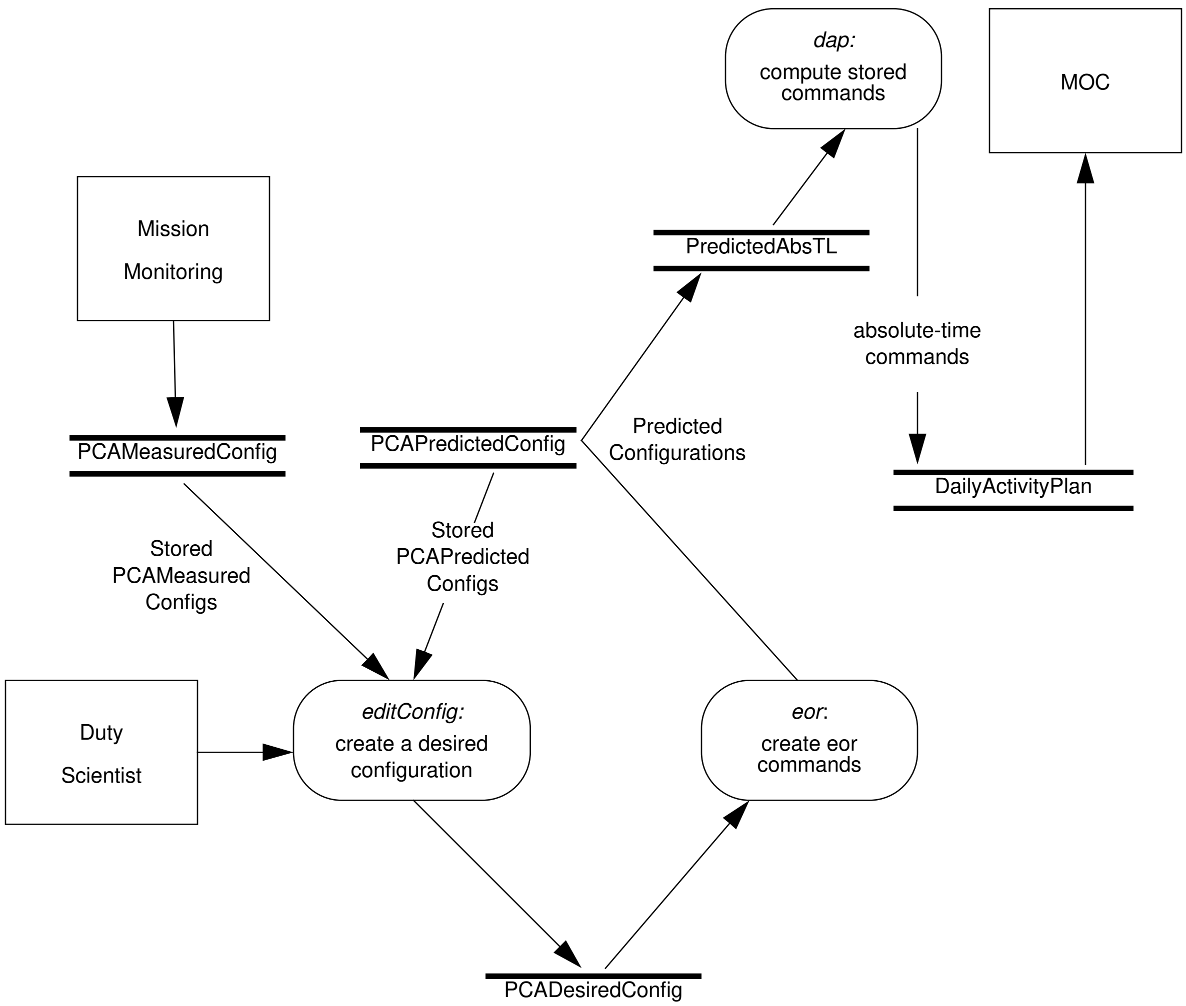


FIGURE 2. PCA Instrument Commanding: Functional Flow (Daily Commanding)



PCADesiredConfig

Synopsis

```
#include <PCADesiredConfig.h>
PCADesiredConfig dc;
```

Description

This is a subclass of `DesiredConfig` for describing the *desired* configuration of observatory subsystems (as opposed to *predicted* configurations) and for generating command scripts. The PCA Instrument Team develops this subclasses for their subsystems that inherit from the base class.

The `DesiredConfig` subclass specifies a value for each commandable parameter of the subsystem. Subsystem parameters may have the *don't care* value when the specific value is unimportant to the configuration or where any previous value is satisfactory. Since the DesiredConfig must remember all desired values including the *don't care* value, the simplest implementation may be to associate a *don't care* boolean flag with each configuration parameter.

Objects of this class are created, for example, when the Guest Observer specifies instrument and data system configurations to the Mission Planning subsystem. The Duty Scientist might modify or add additional configuration details. These objects are also used when the Duty Scientist defines macros, RTSs, EOR loads, and some STOL procedures.

Public Member Functions

`void setConfigurationName (const char* configName);`
Copies the character string `configName` into this object. The configuration name will be used as part of a file name, so please do not make it unreasonably long.

`const char* getConfigurationName() const;`
Returns a pointer to the configuration name that was passed to the `setConfigurationName()` method.

`void setDescription (const char* description);`
Copies the character string `description` into this object. The caller may format the string in any manner but it should be printable by one of the print methods below.

`const char* getDescription () const;`
Returns a pointer to the descriptive text that was passed to the `setDescription()` method.

Virtual Member Functions

`SubsystemId getSubsystemId() const;`
Returns the subsystem identification, an enum `SubsystemId` that has the following values. Only the first four values are allowed for a DesiredConfig; any value in the enum may be used for a PredictedConfig.

`PCA, HEXTE, ASM, EDS, PRAMA, PRAMB, EA1, EA2, EA3, EA4, EA5, EA6, EA7, EA8`

`CommandScript* getCommandScript() const;`
Returns a pointer to the `CommandScript` that puts the subsystem into this desired configuration. Since there is no configuration prediction available, *all* configuration parameters are commanded, except, of course, those with *don't care* values. (Compare this with the `getDeltaCommandScript` method of `PredictedConfig` class.)

If this method cannot produce a valid `CommandScript`, it constructs a `SOCErrObject` to be (1) returned by the `isError` method and (2) inserted into the `CommandScript` itself using its `setErrObject` method. This method also issues an error message on standard error.

This method allocates a `CommandScript` object off the heap using the `new` operator. The caller assumes responsibility for deleting the storage space.

`TelemRate* getTelemRate(const Source& source) const;`
Returns the average telemetry rate of the subsystem when observing the X-ray source `source` in the current configuration. This method allocates a `TelemRate` object off the heap using the `new` operator. The caller assumes responsibility for deleting the storage space.

`PredictedConfig* predictConfig () const;`
Returns a pointer to an object that is the `PredictedConfig` subclass associated with this `DesiredConfig` subclass. This method may return a NULL pointer if it cannot produce a prediction. The predicted configuration object is allocated off the heap and will be deleted by SOC software.

`SOCErrObject* isError() const;`
Returns the current error object for this class, or zero if the preceding operation did not incur an error.

Virtual Print Functions

`void printShort (ostream& ostr) const;`
Prints a *one-line description* of this configuration on the output stream. The format and contents are at the discretion of the Instrument Team, but should include the subsystem name and configuration name.

`void print (ostream& ostr) const;`
Prints a description of this configuration on the output stream, at least as detailed as the one given by `printShort`. The format and contents are at the discretion of the Instrument Team.

`void printLong (ostream& ostr) const;`

Prints a description of this configuration on the output stream, at least as detailed as the one given by `print`. It should include all the configuration values the Instrument Team believes the Duty Scientist will find of primary interest while reviewing observation schedules or during initial scans of incoming science data. The format and contents are at the discretion of the Instrument Team.

Rogue Wave Functions

`PCADesiredConfig ();`
Rogue Wave requires this constructor. The PCA has no requirements for it and does not use it explicitly.

`unsigned binaryStoreSize () const;`
Returns the number of bytes required to store the object.

`void restoreGuts (RWFile&);`
Reads the object's state from a binary file, using class `RWFile`.

`void restoreGuts (RWvistream&);`
Reads the object's state from an input stream.

`void saveGuts (RWFile&) const;`
Writes the object's state to a binary file, using class `RWFile`.

`void saveGuts (RWvostream&) const;`
Writes the object's state to an output stream.

FIGURE 3. PCA Commanding: Main Menu

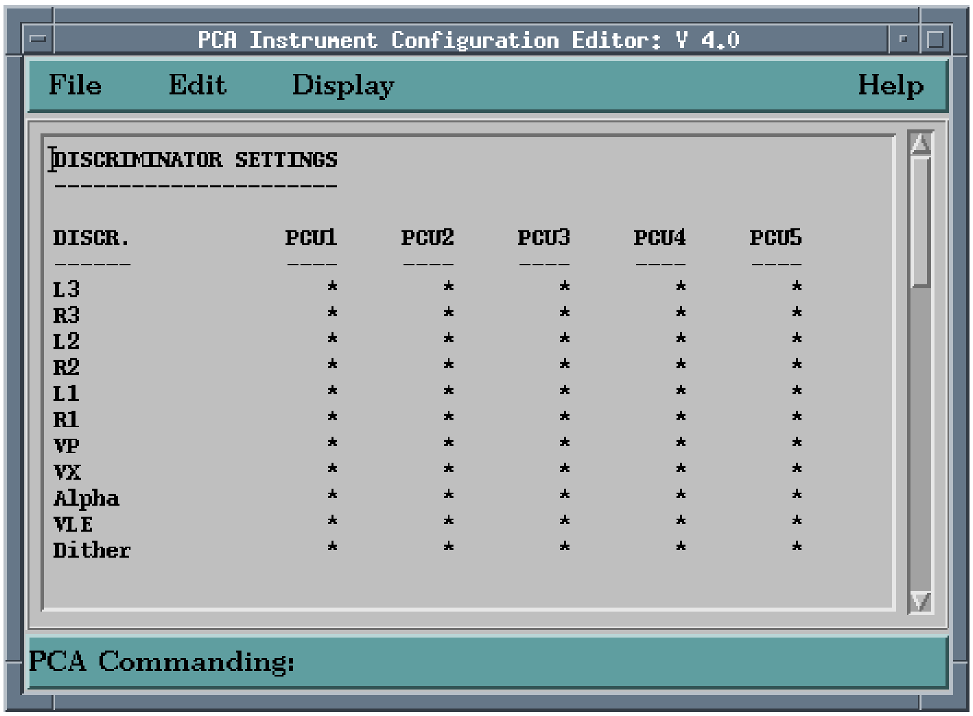


FIGURE 4. PCA Commanding: Discriminator Settings Menu

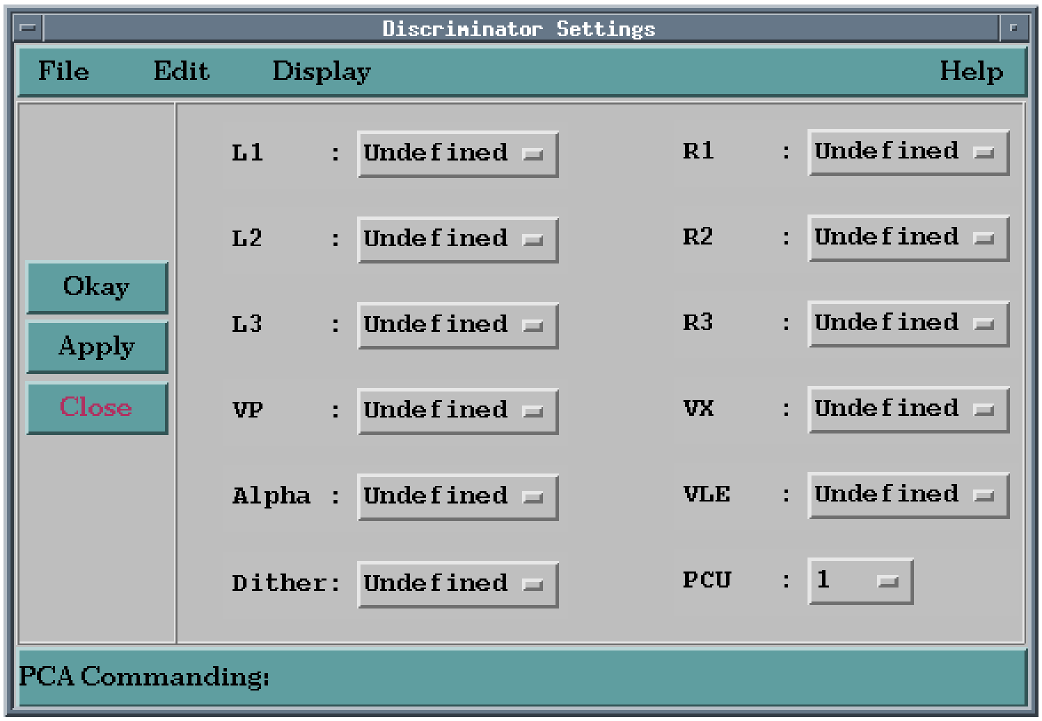


FIGURE 5. PCA Commanding: Test Pulse Generator Settings Menu

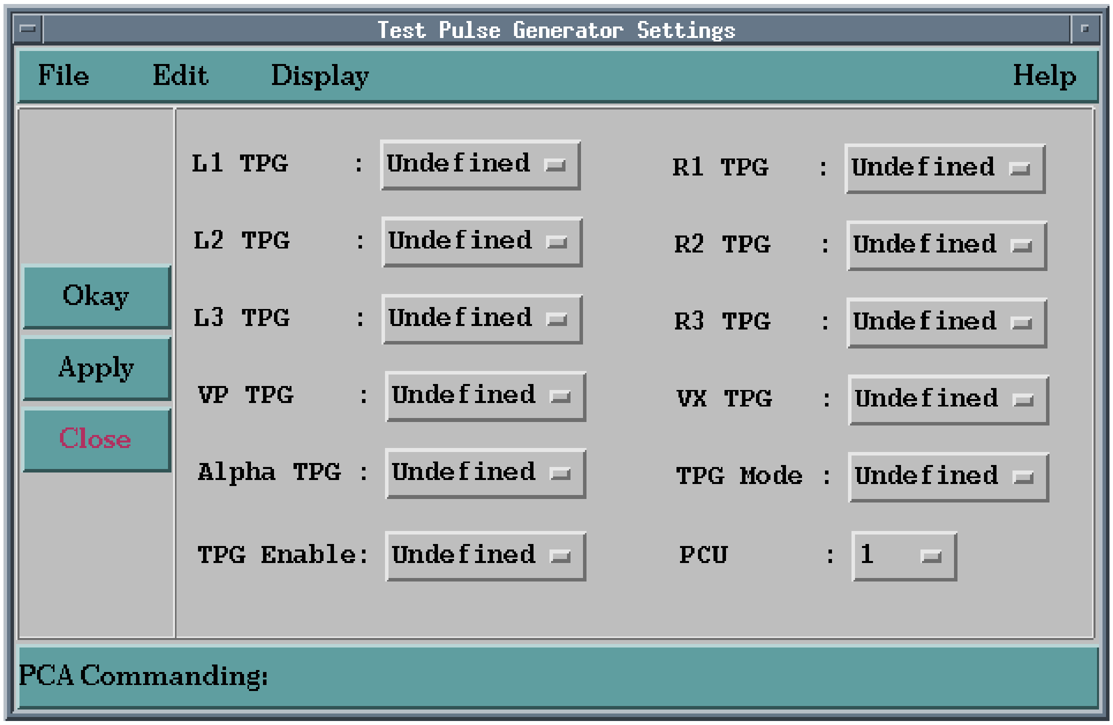
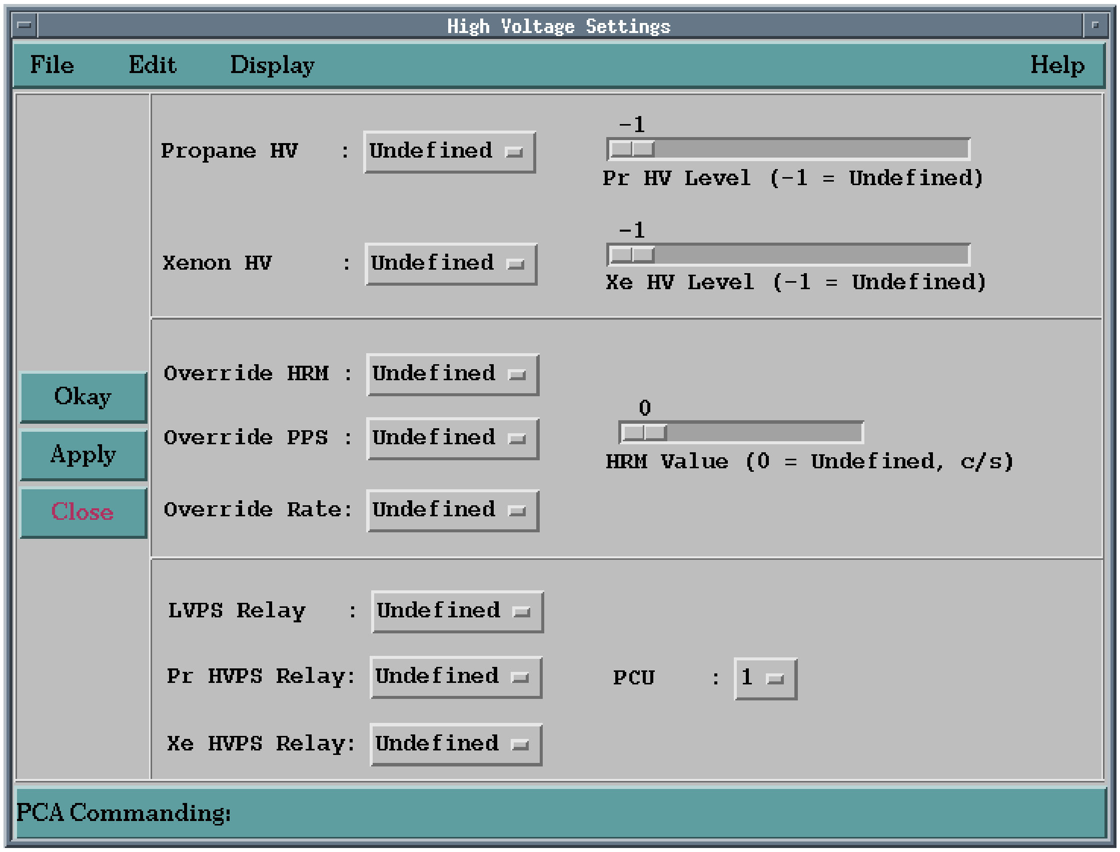


FIGURE 6. PCA Commanding: High Voltage Settings



In addition to these main screens, there are two others. One for the PCA Diagnostic commands and another that presents the configuration summary in which all values can be viewed for a given PCU at a time. This makes verification easier.