LAT Flight Software

LCAT User Manual

Type:	User Manual
Version:	V0-1-0
Author:	S.Maldonado
Created:	11 February 2004
Updated:	4 June 2004
Printed:	4 June 2004

Manual for the LAT FSW Telecommand and Telemetry editor.

Contents

0 Introduction 1
0.0 Overview 1
0.1 Reference Documentation 1
1 Definition Storage 2
1.0 Overview 2
1.1 Containers 2
1.1.0 Subsystem 2
1.1.0.0 APID Range 2
1.2 Definition Files 2
1.2.0 Field Definitions 3
1.2.1 Bitfield Definitions 3
1.2.2 Struct Definitions 3
1.2.3 Packet Definitions 3
1.2.4 Ranges (Telecommand) 3
1.2.5 Enumerations (Telecommand) 4
1.2.5.0 Enumeration Members 4
1.2.6 Discrete Conversions (Telemetry) 4
1.2.7 Analog Conversions (Telemetry) 4
1.2.8 Limit Sets (Telemetry) 4
2 The LCAT Application 5
2.0 Application Flow 5
2.1 Getting Started 6
2.1.0 The Cat Directory 6
2.1.1 Launch LCAT 7
2.1.2 The Main Window 7
2.2 Containers 8
2.2.0 Creating a New Container 8
2.2.1 Editing Container Contents 9
2.2.2 Editing the Subsystem Object 10
2.3 Parameter Definitions 11
2.3.0 Creating Fields 11
2.3.1 Creating Bitfields 12
2.3.2 Creating Structs 13
2.3.3 Creating Packets 14
2.3.3.0 Telecommand Packets 14
2.3.3.1 Telemetry Packets 15
2.4 Constraint Definitions 16
2.4.0 Creating Ranges 16
2.4.1 Creating Enumerations 17
2.4.1.0 Adding Enumeration Values 18
2.4.2 Creating Discrete Conversions 19
2.4.2.0 Adding Discrete Conversion Values 20
2.4.3 Creating Analog Conversions 21

2.4.4 Creating Limit Sets 22
2.5 Container Validation 23
2.6 Adding ITOS Mnemonics 24
2.7 Generating ITOS 25
2.8 Generating Source Code 25
2.9 LCAT Command Line Utilities 25
2.9.0 lcat validate 26
2.9.1 lcat generate 26
2.9.1.0 source 26
2.9.1.1 itos 26
2.9.1.2 dependencies 26
2.9.1.3 xml 26
3 LCAT Products 27
3.0 ITOS 27
3.0.0 ITOS Versions 27
3.1 C Source Code 27
3.1.0 Packet Structs 27
3.1.1 ITC Routing Tables 28
3.1.2 Callback Function Prototypes 28
4 Installing LCAT 29
4.0 Prerequisites 29
4.0.0 Prerequisite Enumeration 29
4.0.0.0 Full GUI LCAT 29
4.0.1 Practical Experience With Prerequisites 29
4.1 Summary 30

Figures

Figure 1 LCAT Main Window 7
Figure 2 New Container Dialog 8
Figure 3 Container Editing 9
Figure 4 Subsystem Editor 10
Figure 5 Field Editor 11
Figure 6 Bitfield Editor 12
Figure 7 Struct Editor 13
Figure 8 Telecommand Packet Editor 14
Figure 9 Telemetry Packet Editor 15
Figure 10 Range Editor 16
Figure 11 Enumeration Editor 17
Figure 12 Enumeration Value Editor 18
Figure 13 Discrete Conversion Editor 19
Figure 14 Discrete Conversion Value Editor 20
Figure 15 Analog Conversion Editor 21
Figure 16 Limit Editor 22
Figure 17 Container Validation 23
Figure 18 ITOS Mnemonic Editor 24
Figure 19 Generating ITOS 25

Figure 20 Section of code in `sipQt.h` before patch 30
Figure 21 Section of code in `sipQt.h` after patch 30

0 Introduction

0.0 Overview

LCAT, the LAT Command and Telemetry tool, is a graphical application used to create, edit, and maintain telecommand and telemetry definitions for the LAT instrument. It is intended for use by the LAT flight software group throughout the development process. Consequently, the application is integrated with the existing flight software development environment, CMX. LCAT relies on the CMX code management framework to organize all command and telemetry definition files into self-contained software subsystems, known as packages.

The primary data storage format is XML, a standard that provides for a hierarchical organization of information, and lends itself well to describing the structural relationships inherent in command and telemetry definitions. The widespread use of the XML specification facilitates portability of definitions between the flight software group and other LAT organizations, such as Integration and Test (I&T) and the Instrument Science Operations Center (ISOC).

LCAT has the ability to translate XML definitions to other known formats. One such format is ITOS (Integrated Test and Operations System), which is accpeted by the spacecraft vendor's command and control system, AstroRT, and by the GLAST Mission Operations Center (MOC).

Flight source code (C structs, function prototypes, stubs) can also be generated directly from the XML definitions. This ensures a consistent implementation of command interpretation and telemetry packet construction throughout all LAT software subsystems.

0.1 Reference Documentation

1. "GLAST Database Format Control Document", NASA Goddard Space Flight Center

2. "GLAST 1553 Bus Protocol Document", Spectrum Astro

3. TD-02659 , "LAT Flight Software Telecommand and Telemetry Formats", by Dan Wood

1 Definition Storage

1.0 Overview

The primary storage mechanism for LCAT definitions are XML files. All information is organized into a strict hierarchy, which is enforced by the application's XML parser and the associated Document Type Definition (DTD) file. Command and telemetry definitions are stored in multiple "definition" files. A single top level "container" file holds a reference to each definition file and allows for the sharing or reuse of data content between them.

All LCAT container files use the file extension ".catc". All definition files use the file extenstion ".cath".

1.1 Containers

A container is the top level file that describes a complete set of commands and telemetry for a software subsystem. This container's dataset consists of a single subsystem description, a list of files (definitions), and lists of command and telemetry that are valid for the subsystem.

1.1.0 Subsystem

A subsystem object has a one-to-one relationship with a container and is typically named after the CMX package where the files reside.

1.1.0.0 APID Range

A subsystem object describes the Application ID, or APID range for which all its associated telecommands and telemetry are valid. In LCAT all APID ranges are specified as decimal values.

1.2 Definition Files

Definition files contain the actual definitions of packets, parameters, and constraints. These definitions can be reused throughout the container.

A separate definition file exists for type as follows:

- Fields (cmd and tlm)
- Bitfields (cmd and tlm)
- Structs (cmd and tlm)
- Packets (cmd and tlm)
- Ranges (cmd only)
- Enumerations
- Discrete Conversions
- Analog Conversions
- Limit Sets

1.2.0 Field Definitions

Fields are wrappers for the integral data types:

- (unsigned) char
- (unsigned) short
- (unsigned) int
- (unsigned) long long
- float
- double
- string

1.2.1 Bitfield Definitions

Bitfields definitions are wrappers for C bitfields and use LCAT field definitions as member values.

1.2.2 Struct Definitions

Structs definitions are wrappers for C structs and use LCAT field, bitfield, and struct definitions as member values.

1.2.3 Packet Definitions

A packet is identified by an APID, and can be referenced by a unique mnemonic. Packet definitions contain parameter references and must be unique within the entire LAT command set.

Packet definitions use LCAT field, bitfield, and struct definitions as member values.

1.2.4 Ranges (Telecommand)

A range constraint defines a field value's upper and lower bounds such that lower_bound <= field_value <= upper_bound. A range must have a name unique within a container.

1.2.5 Enumerations (Telecommand)

An enumeration defines the discrete value set for a field. Each set consists of members describing a valid name/value pair. An enumeration name must be unique within a container.

1.2.5.0 Enumeration Members

Enumeration members define the fixed value and value name pair used to interpret a field's value. Member names should be unique within each enumeration.

1.2.6 Discrete Conversions (Telemetry)

Discrete conversions transform a range of numeric values into a set of text strings for a field value.

1.2.7 Analog Conversions (Telemetry)

Analog conversions convert integer numbers of counts into floating point values in engineering units. Coefficients are defined for 8^{th} order polynomials.

1.2.8 Limit Sets (Telemetry)

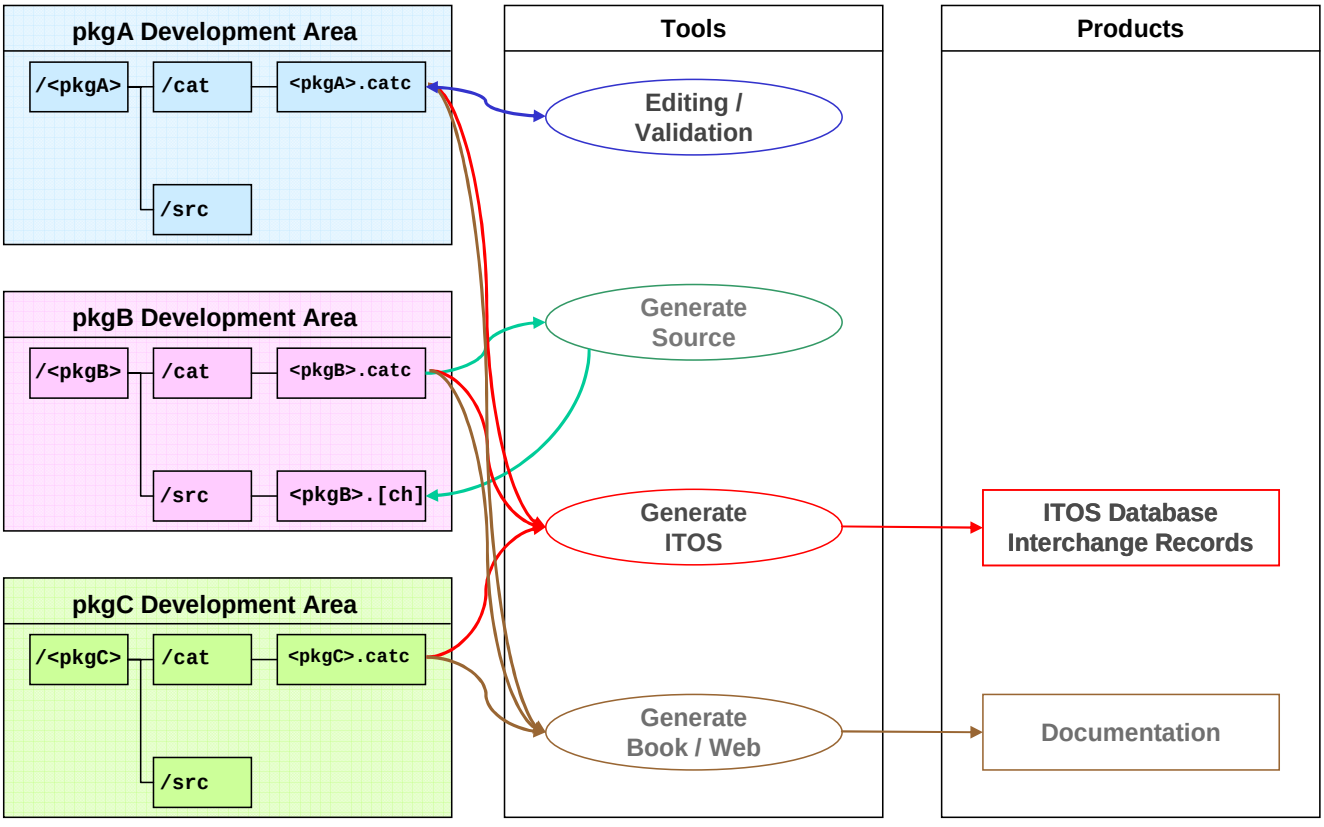
Limit sets define thresholds for field values and consist of yellow and red ranges.

2 The LCAT Application

The LCAT application provides a main workspace and multiple data entry dialogs for definition editing. LCAT was developed using Python and Qt. It uses a validating xml parser with a Document Type Definition (DTD) file to ensure adherence to the hierarchy and enforce associated rules.

2.0 Application Flow

The following chart describes the process involved in running LCAT. All associated files are stored in the package tree and products are generated upon completion of the editing phase.



2.1 Getting Started

LCAT is itself a CMX package. In order to run the application, the user must have an active CMX environment in place. To start using LCAT, first create a cat directory.

2.1.0 The Cat Directory

Create a new subdirectory in the CMX package tree called “cat”. This is where all LCAT files and some output products will be stored.

- CMX Package
  - PKG
  - Visual
  - **cat**
  - cmt
  - doc
  - ptd
  - sdf
  - src

2.1.1 Launch LCAT

Start a CMX session, cd to the newly created “cat” directory, and enter “lcatui” to launch the main application window.

2.1.2 The Main Window

The main window consists of a toolbox on the left, a tabbed workspace in the middle, a summary container view on the right, and a toolbar on top. The toolbox displays available objects that can be added to the workspace. The workspace displays lists for active objects. The container view displays the contents of all objects in the container in a hierarchical representation.

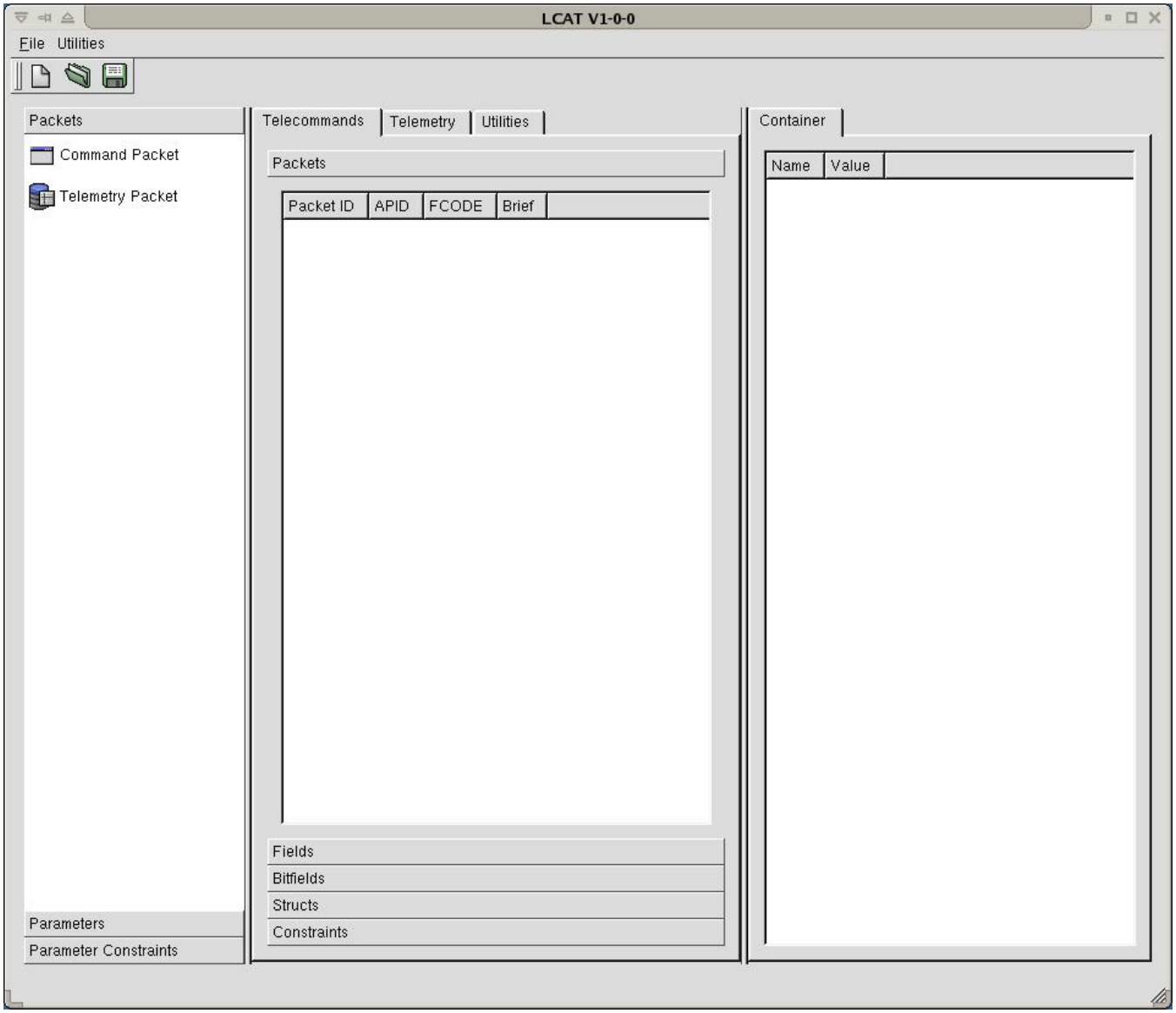


Figure 1 LCAT Main Window

2.2 Containers

2.2.0 Creating a New Container

Create a new container file by selecting File->New from the toolbar. Enter the path and filename of the new container in the dialog box.

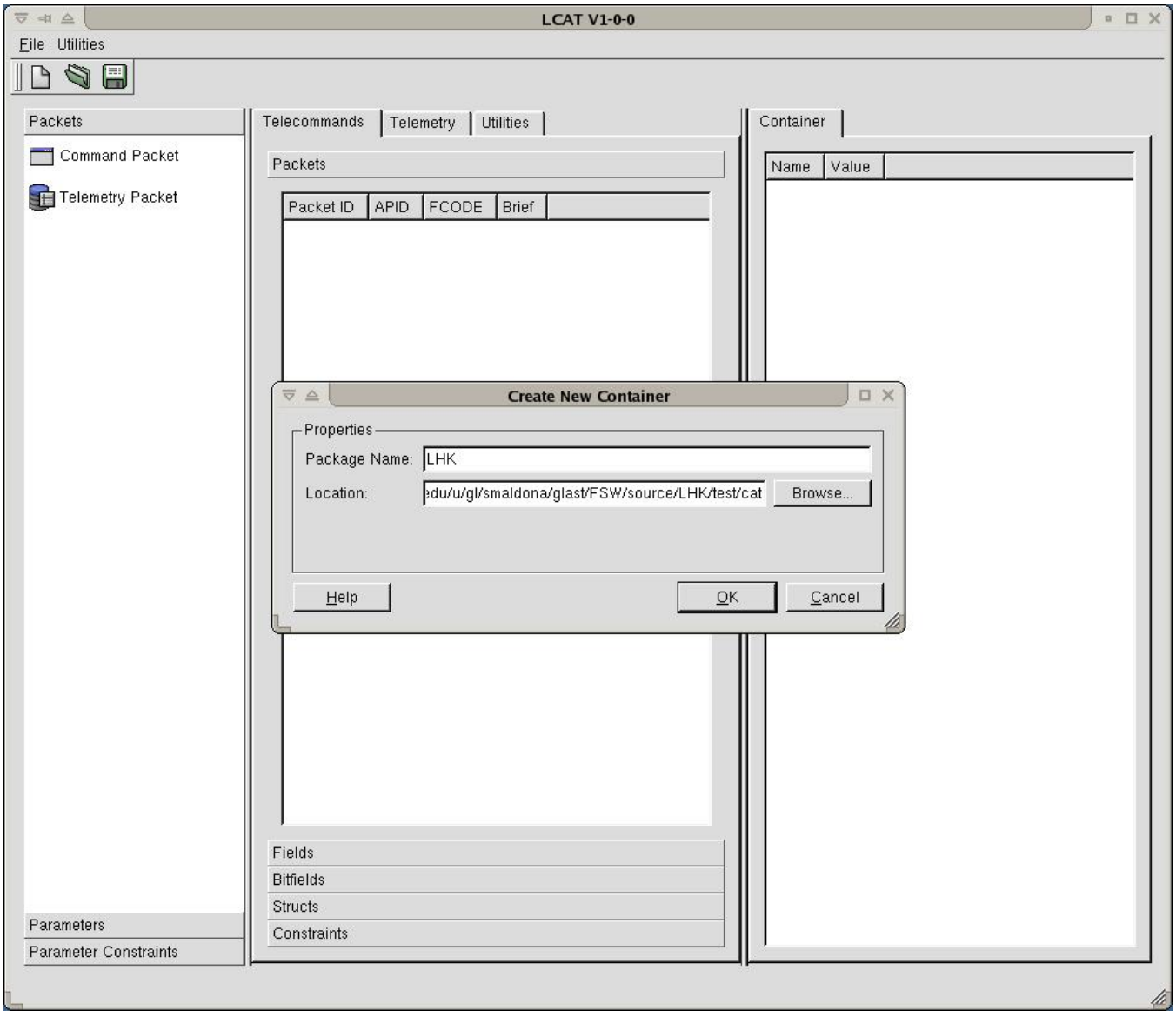


Figure 2 New Container Dialog

Container files should be named after the package in which they reside.

2.2.1 Editing Container Contents

Once a new, container is created, items can be added using the toolbox on the left, or by right-clicking in the associated window to bring up a context menu.

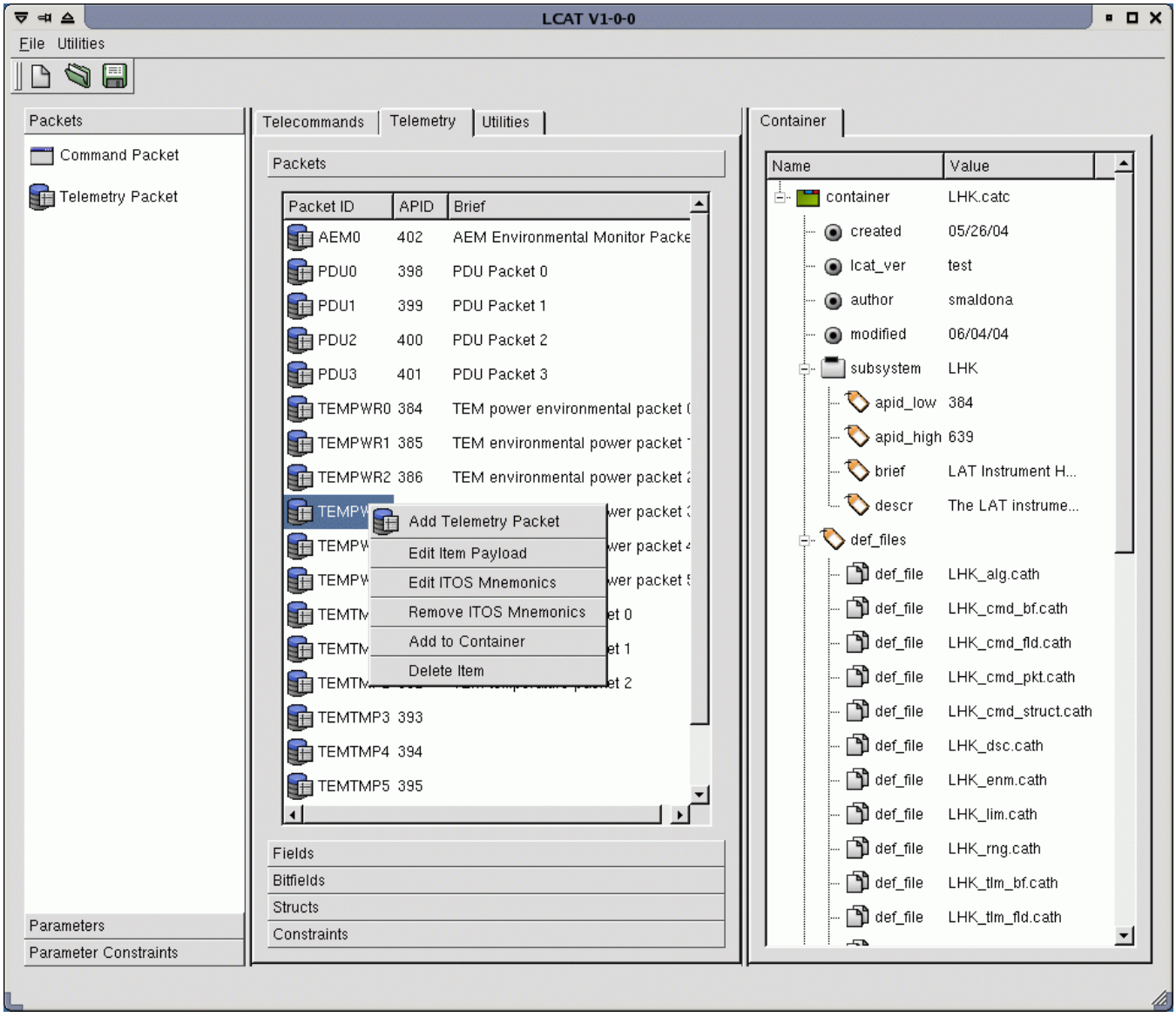


Figure 3 Container Editing

2.2.2 Editing the Subsystem Object

Select the “Utilities” tab and press the “Subsystem Properties” button to open the subsystem editor dialog. Populate the fields with the subsystem information then press OK to accept. You now have a container ready to receive items.

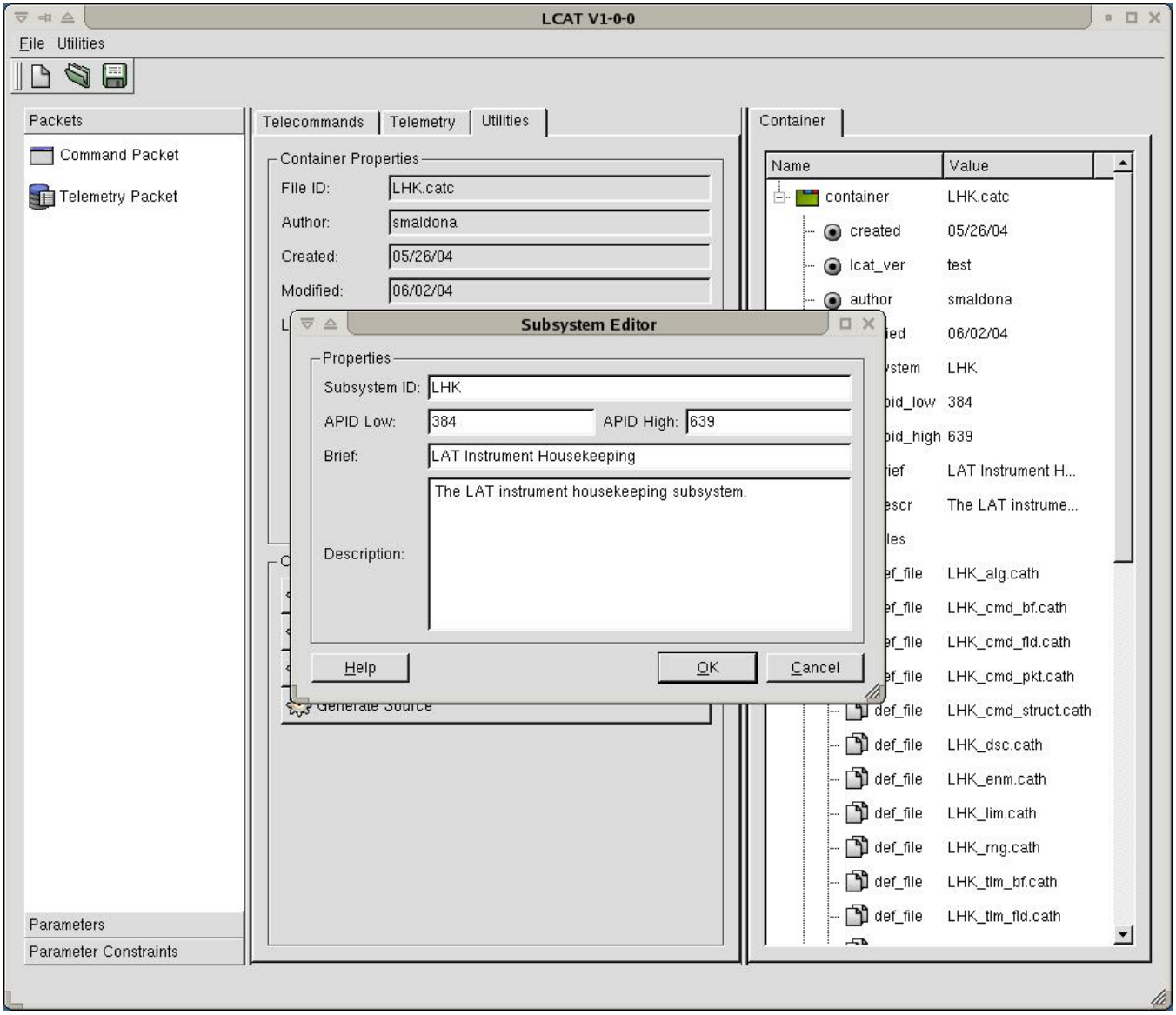


Figure 4 Subsystem Editor

2.3 Parameter Definitions

Parameters are objects that can be used in a packet. Before a packet is created, parameters need to be defined.

2.3.0 Creating Fields

A field definition specifies a data type, length, and some descriptive text. If a field is being defined for use in a bitfield, manually set the bit length to the appropriate value.

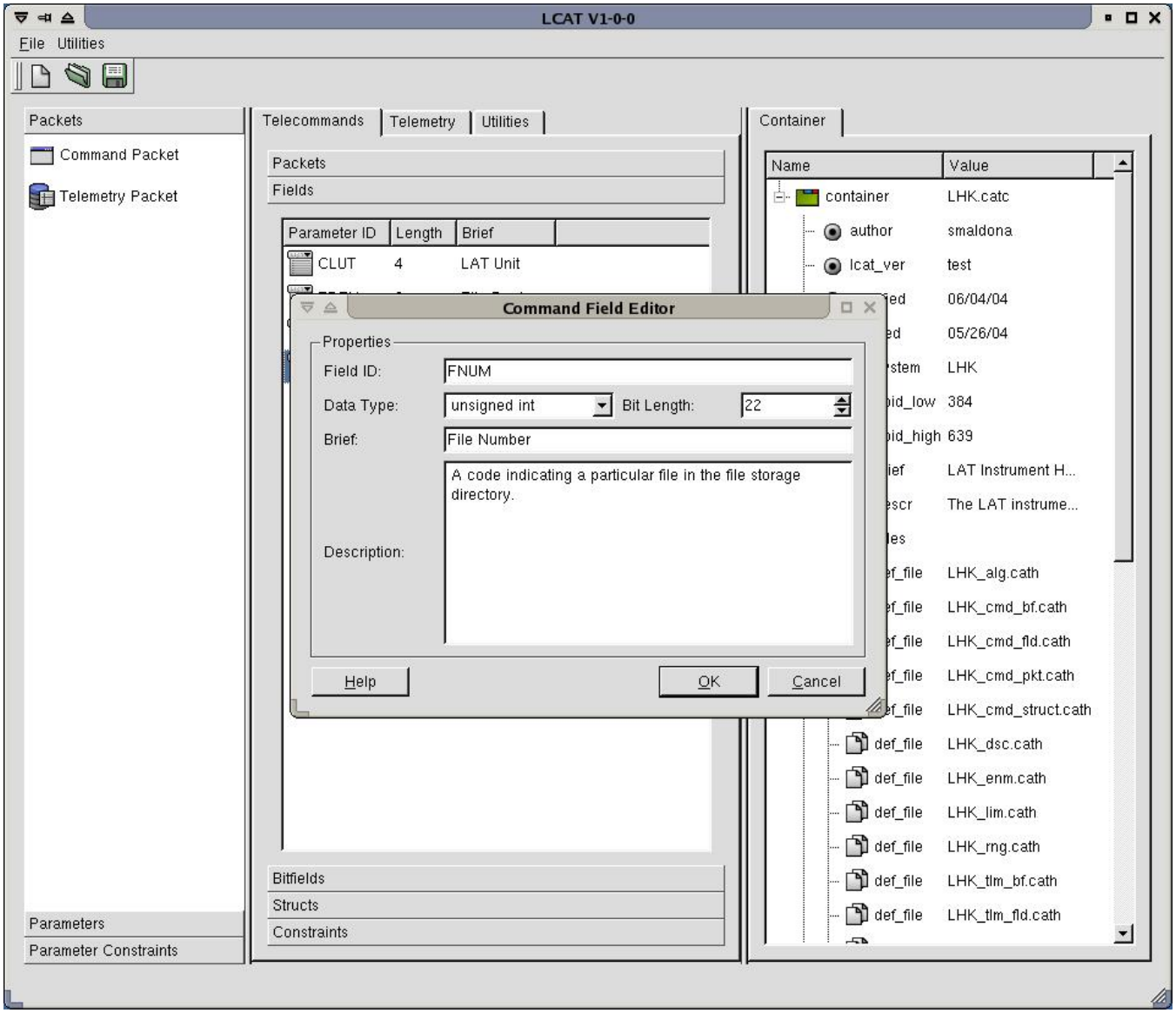


Figure 5 Field Editor

2.3.1 Creating Bitfields

Bitfields specify a data type and descriptive text. To add new fields, select the type using the pull down selector in the “Field” column. Enter an instance name in the “Name” column. Specify any constraints using the appropriate pull down selectors.

Pressing the “Apply” button will calculate and display member positions and alignment padding.

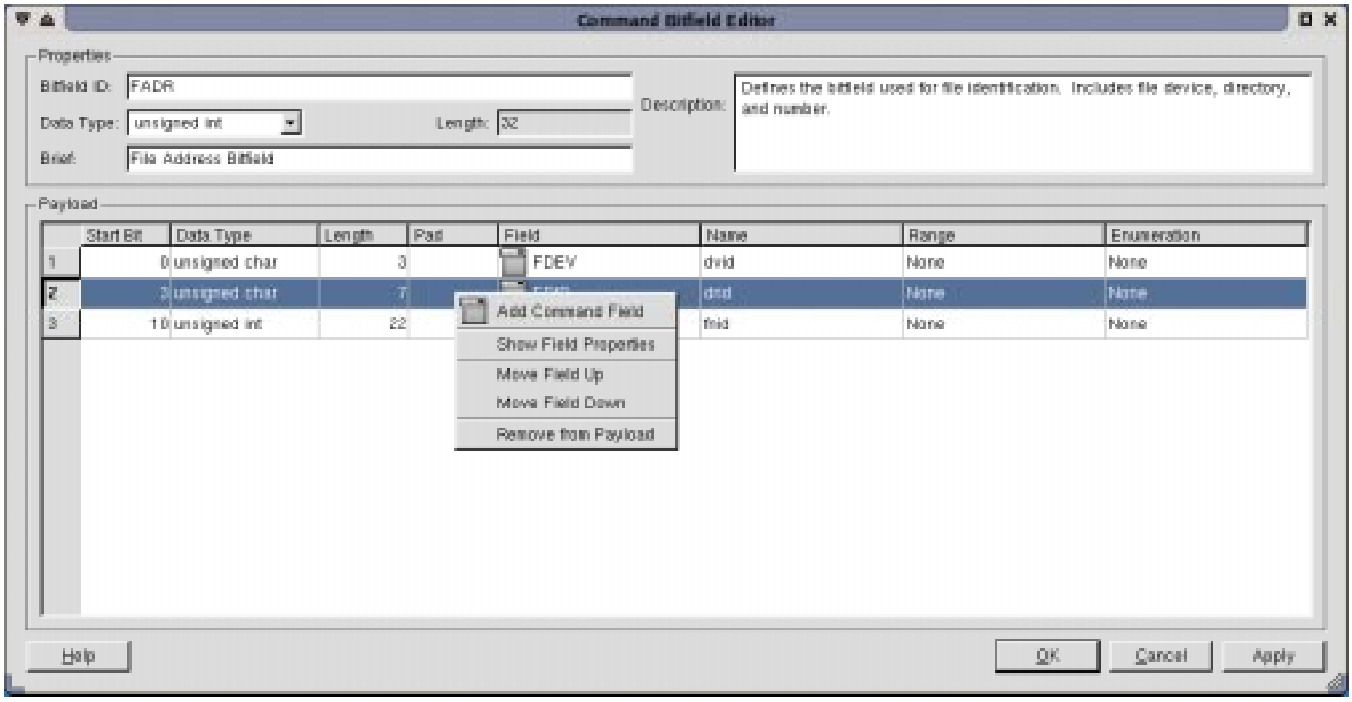


Figure 6 Bitfield Editor

The “Field” and “Name” column values are analogous to C bitfield data type and instance name declarations.

2.3.2 Creating Structs

Structs use fields, bitfields and other structs as member values and also specify some descriptive text. Add member values and provide instance names in the “Name” column. Specify constraints on integral types only.

Arrays can be specified here by checking the “Array” box and selecting the array length in the “Count” column. Note that constraints specified for arrays will be applied to each array index.

Pressing the “Apply” button will calculate and display member positions and alignment padding.

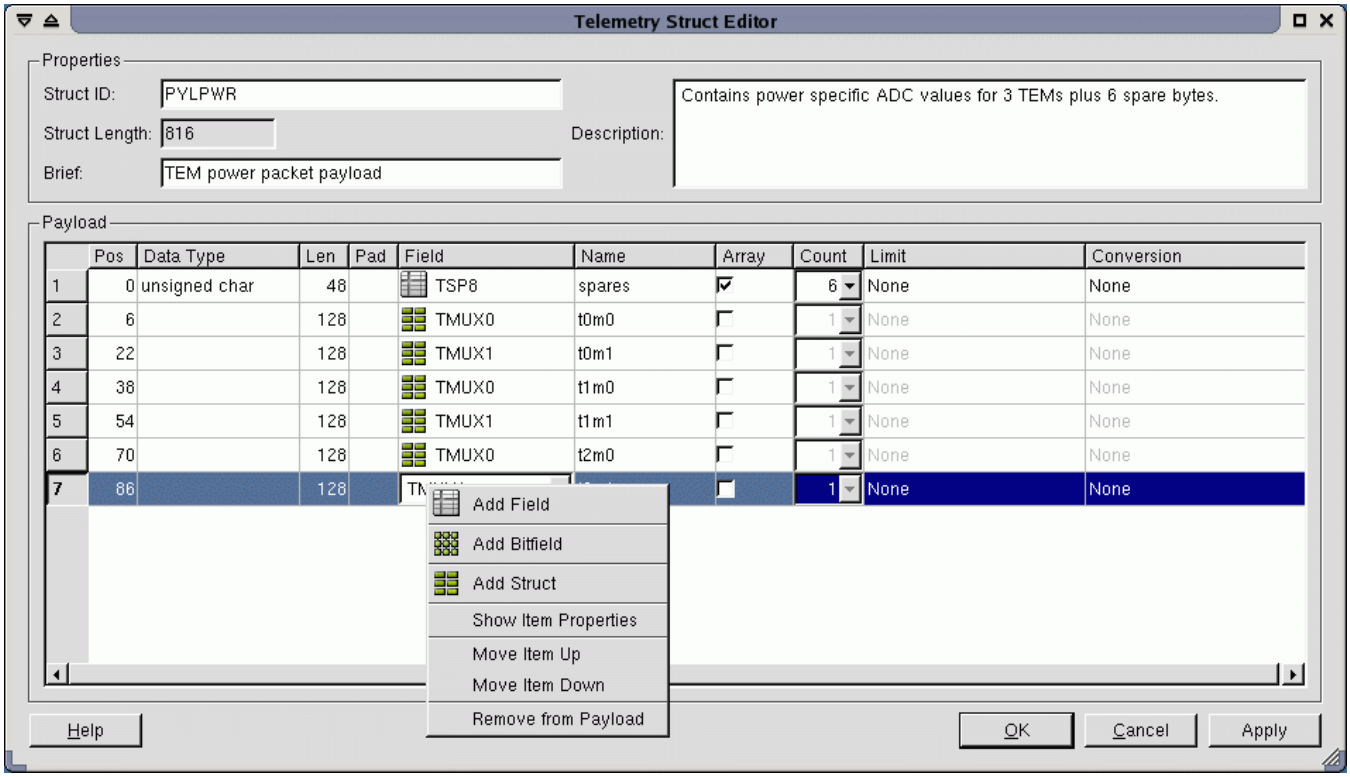


Figure 7 Struct Editor

The “Field” and “Name” column values are analogous to C struct data type and instance name declarations.

2.3.3 Creating Packets

2.3.3.0 Telecommand Packets

Create telecommand packets using the pre-assigned APID ranges and must specify an integer function code. Edit packets in the same manner as structs.

Note the addition of bit padding in the “Pad” column for misaligned payloads. Pressing the “Apply” button will calculate and display member positions and alignment padding. All packet and parameter length values are specified in bits.

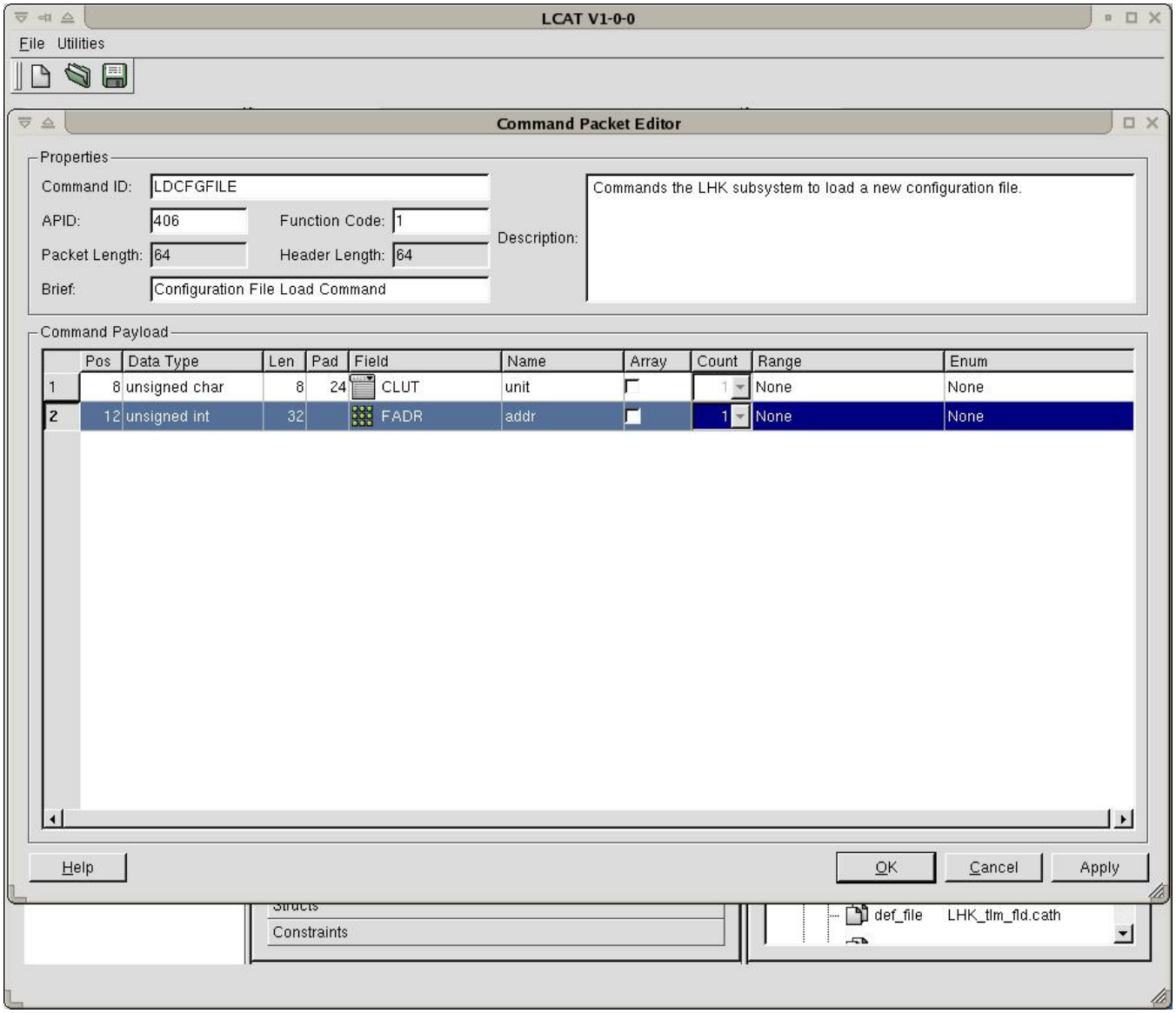


Figure 8 Telecommand Packet Editor

APIDs must be specified as integer values.

2.3.3.1 Telemetry Packets

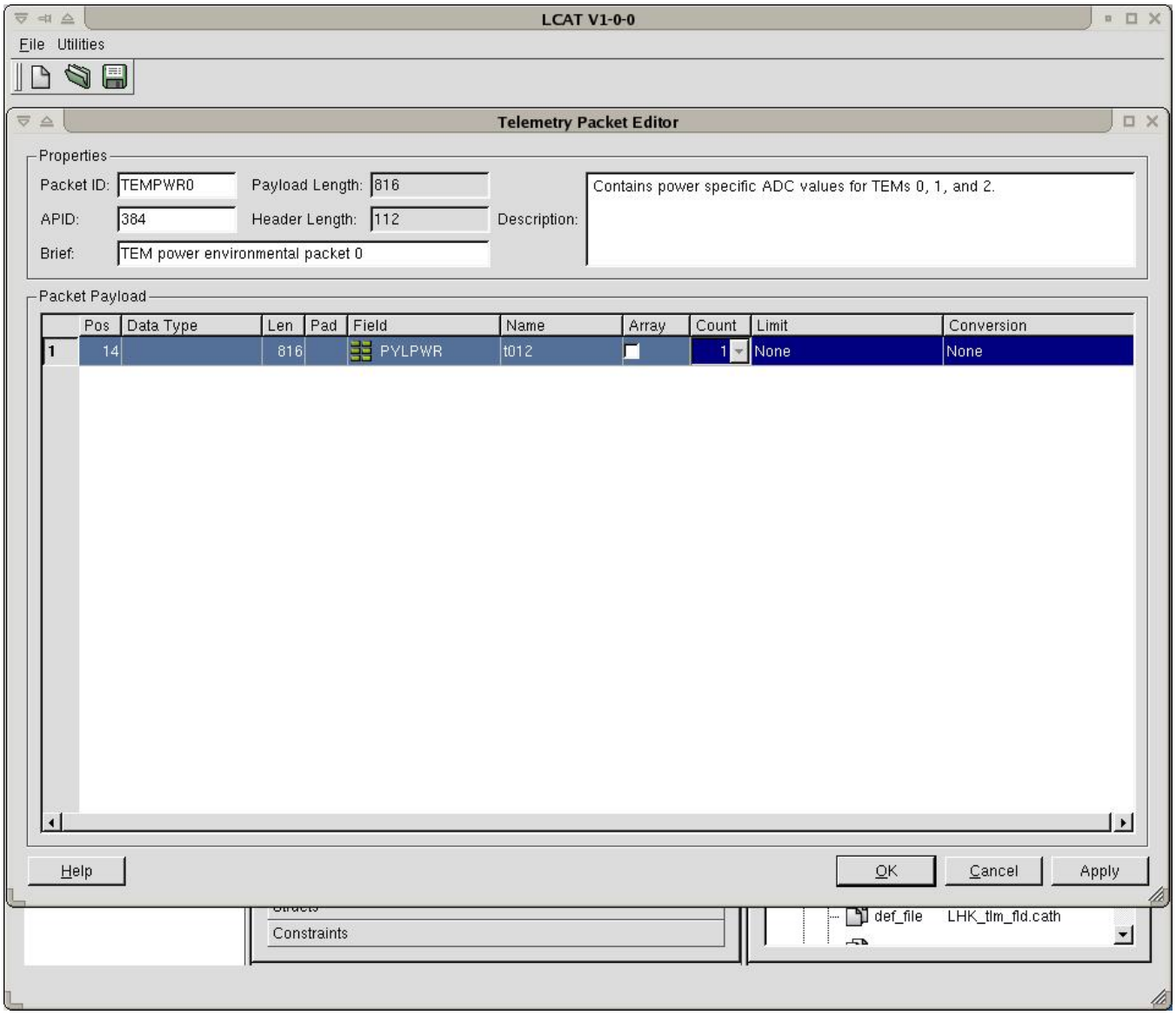


Figure 9 Telemetry Packet Editor

2.4 Constraint Definitions

2.4.0 Creating Ranges

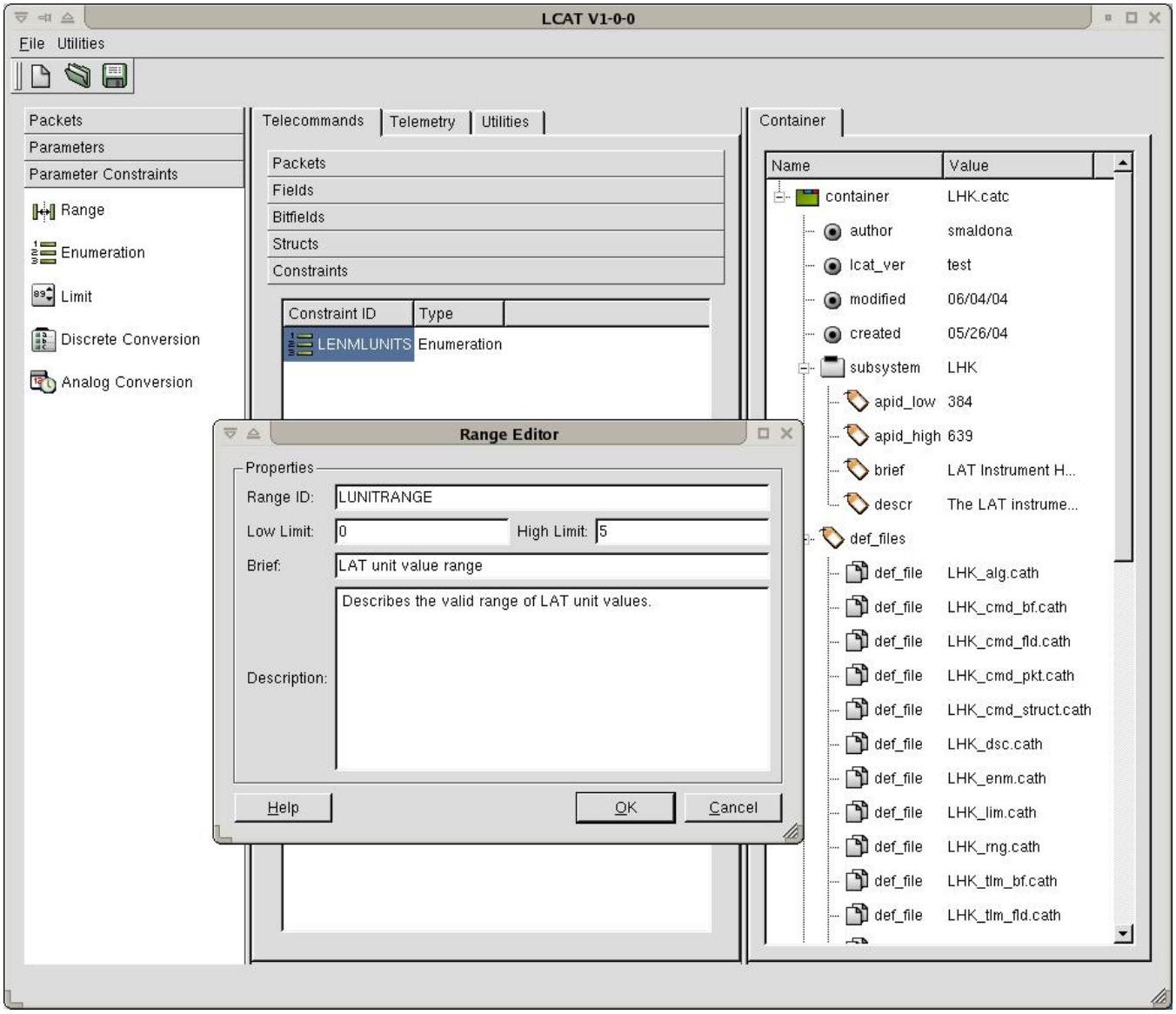


Figure 10 Range Editor

2.4.1 Creating Enumerations

Create enumerations and add member values.

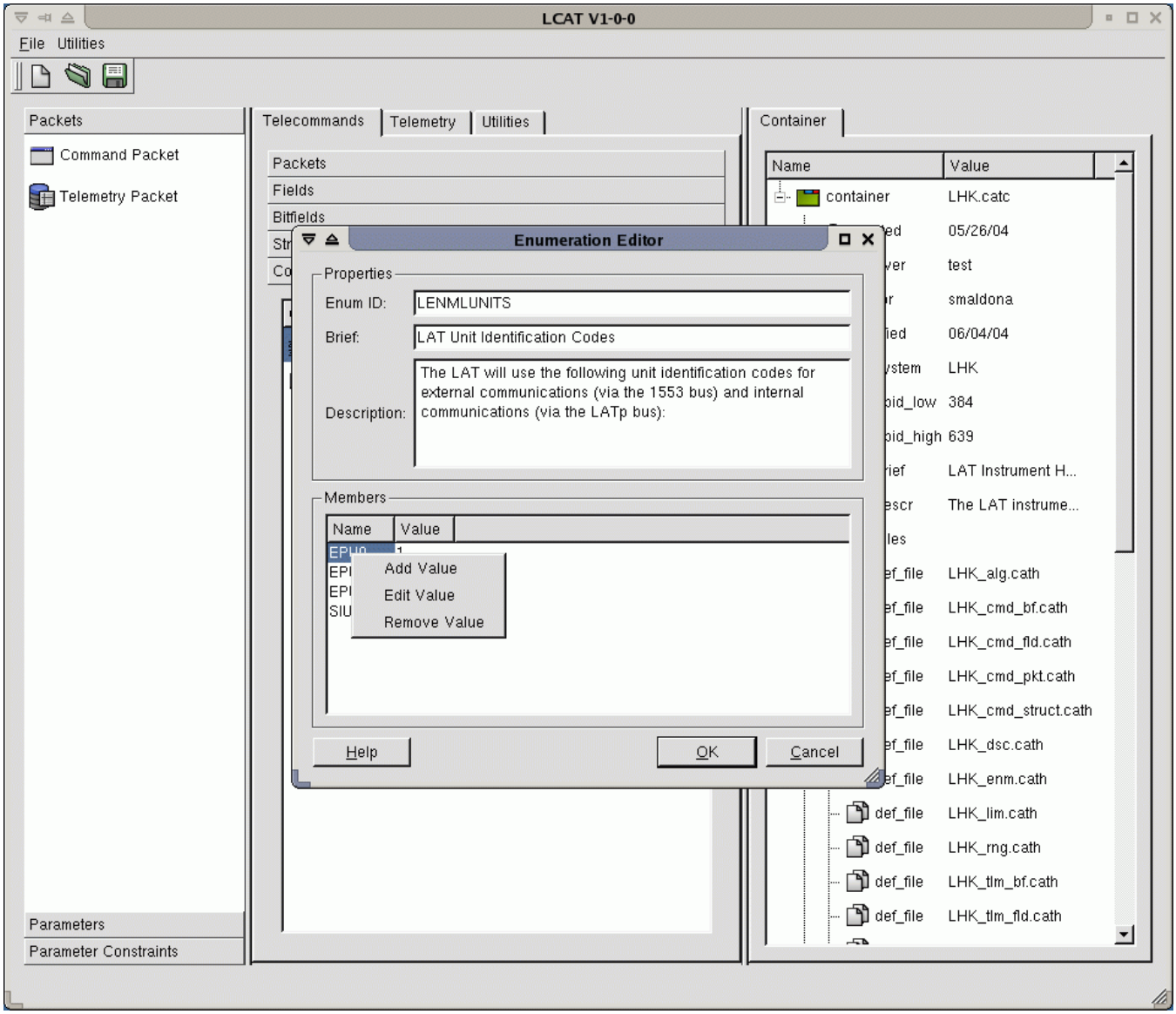


Figure 11 Enumeration Editor

2.4.1.0 Adding Enumeration Values

Specify fixed values as integers.

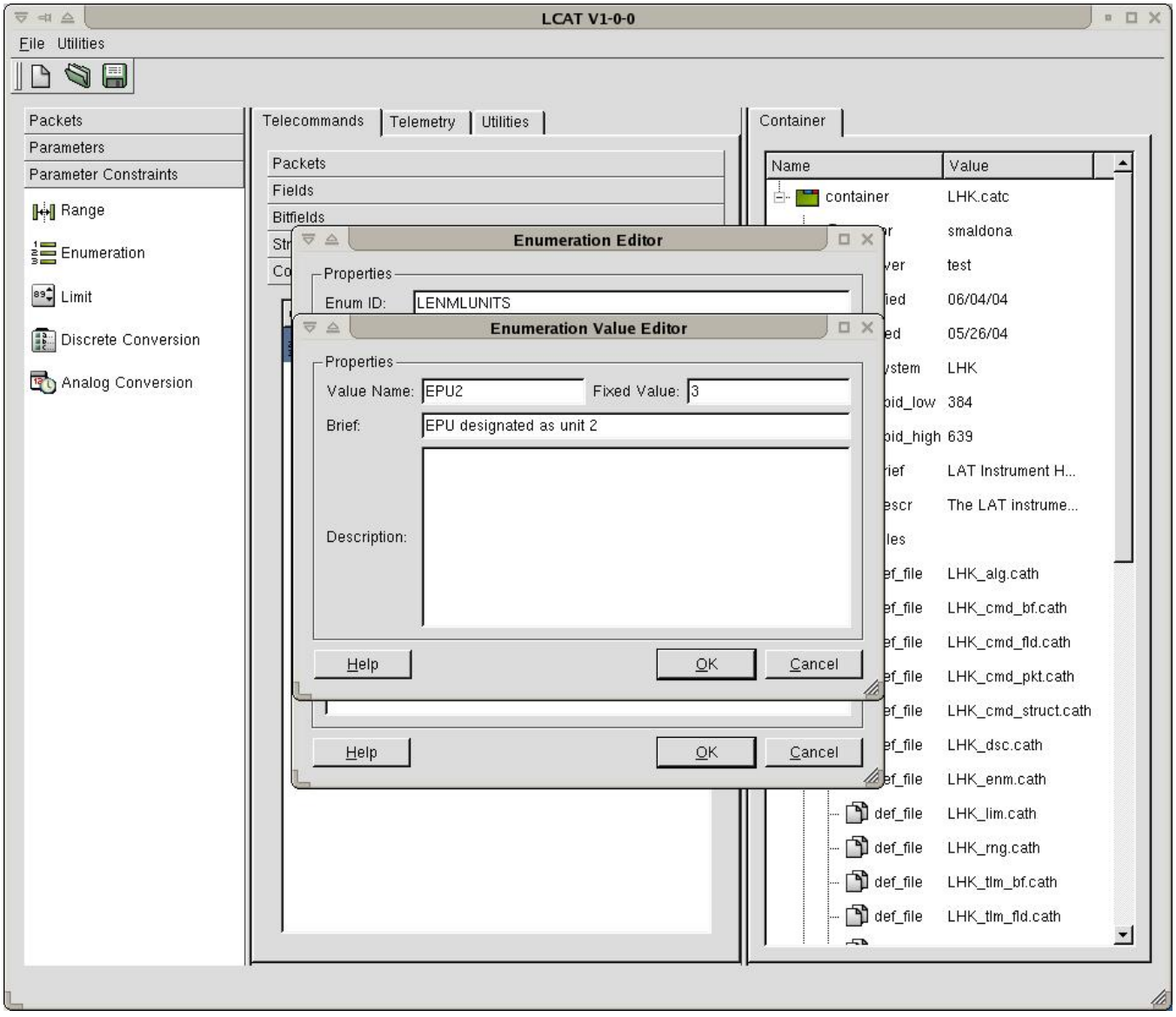


Figure 12 Enumeration Value Editor

2.4.2 Creating Discrete Conversions

Create discrete conversions and add member values.

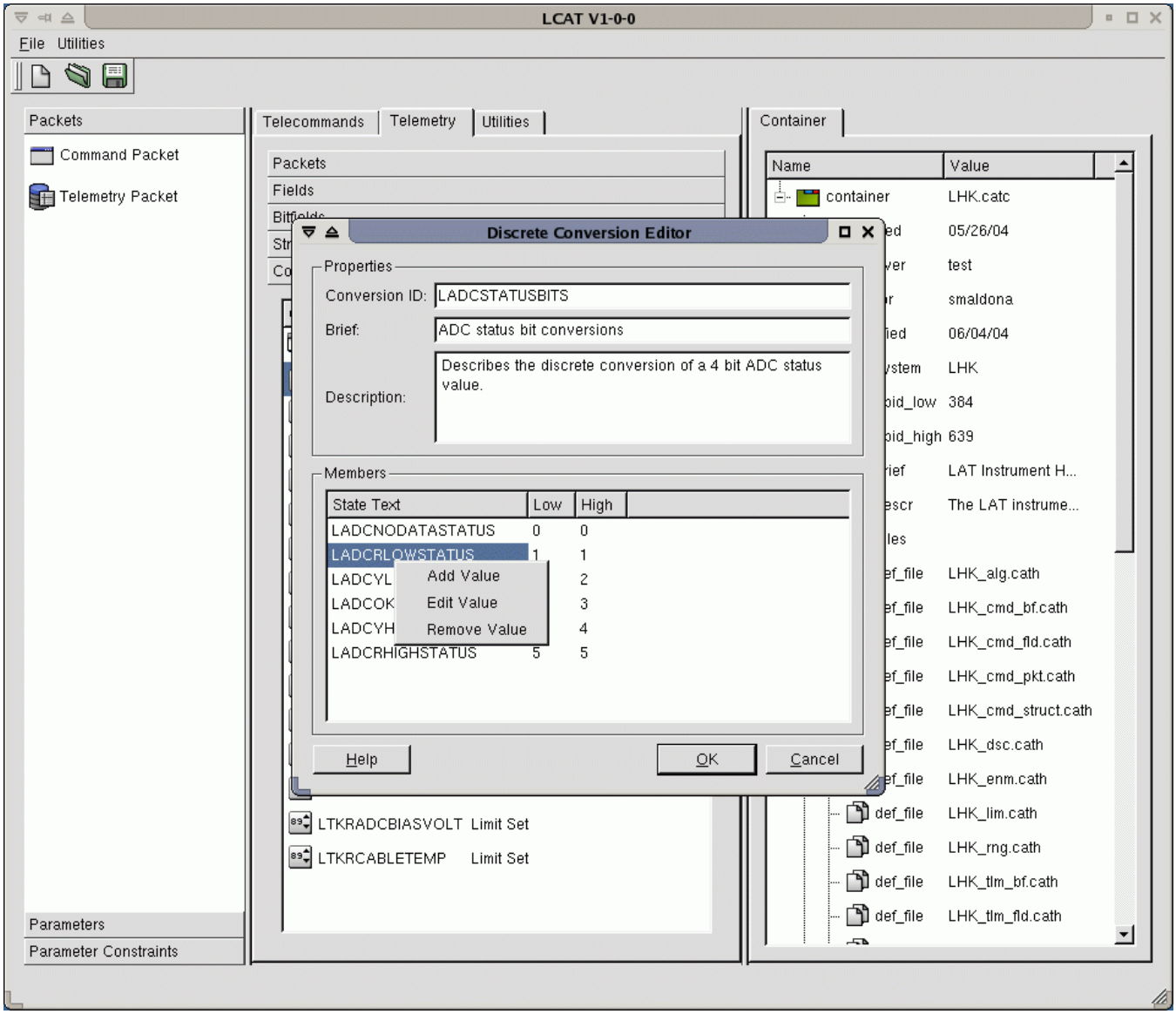


Figure 13 Discrete Conversion Editor

2.4.2.0 Adding Discrete Conversion Values

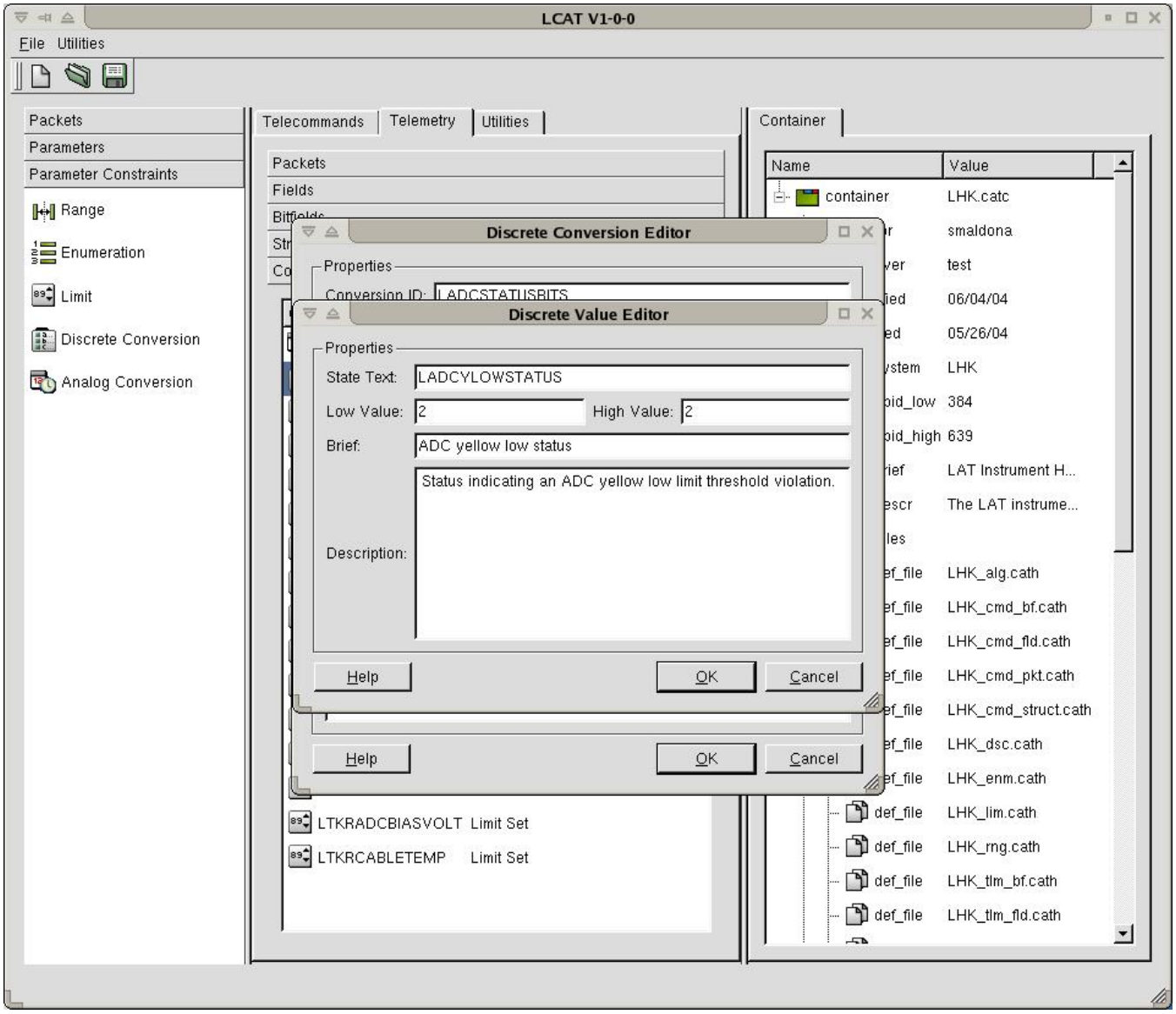


Figure 14 Discrete Conversion Value Editor

2.4.3 Creating Analog Conversions

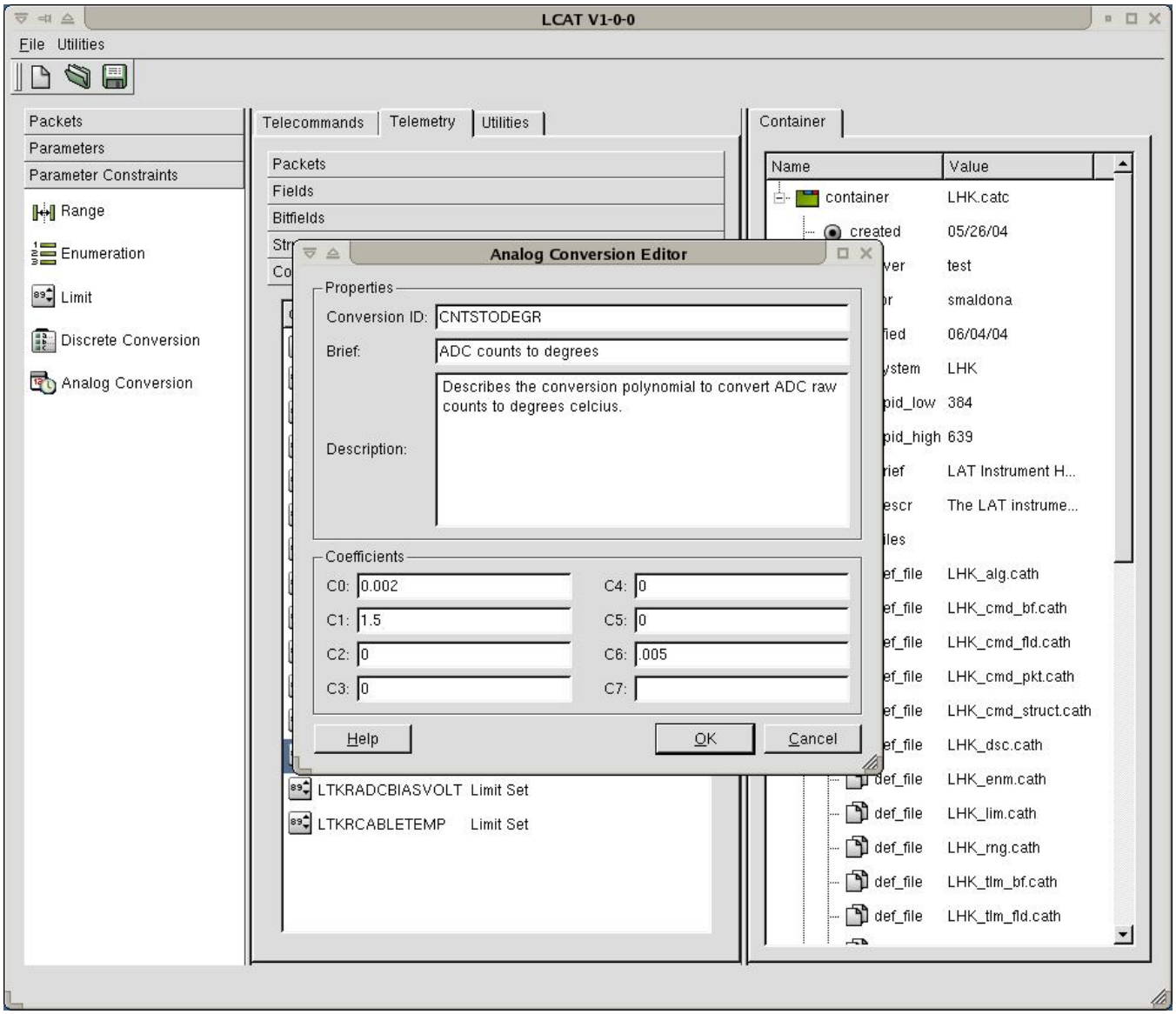


Figure 15 Analog Conversion Editor

2.4.4 Creating Limit Sets

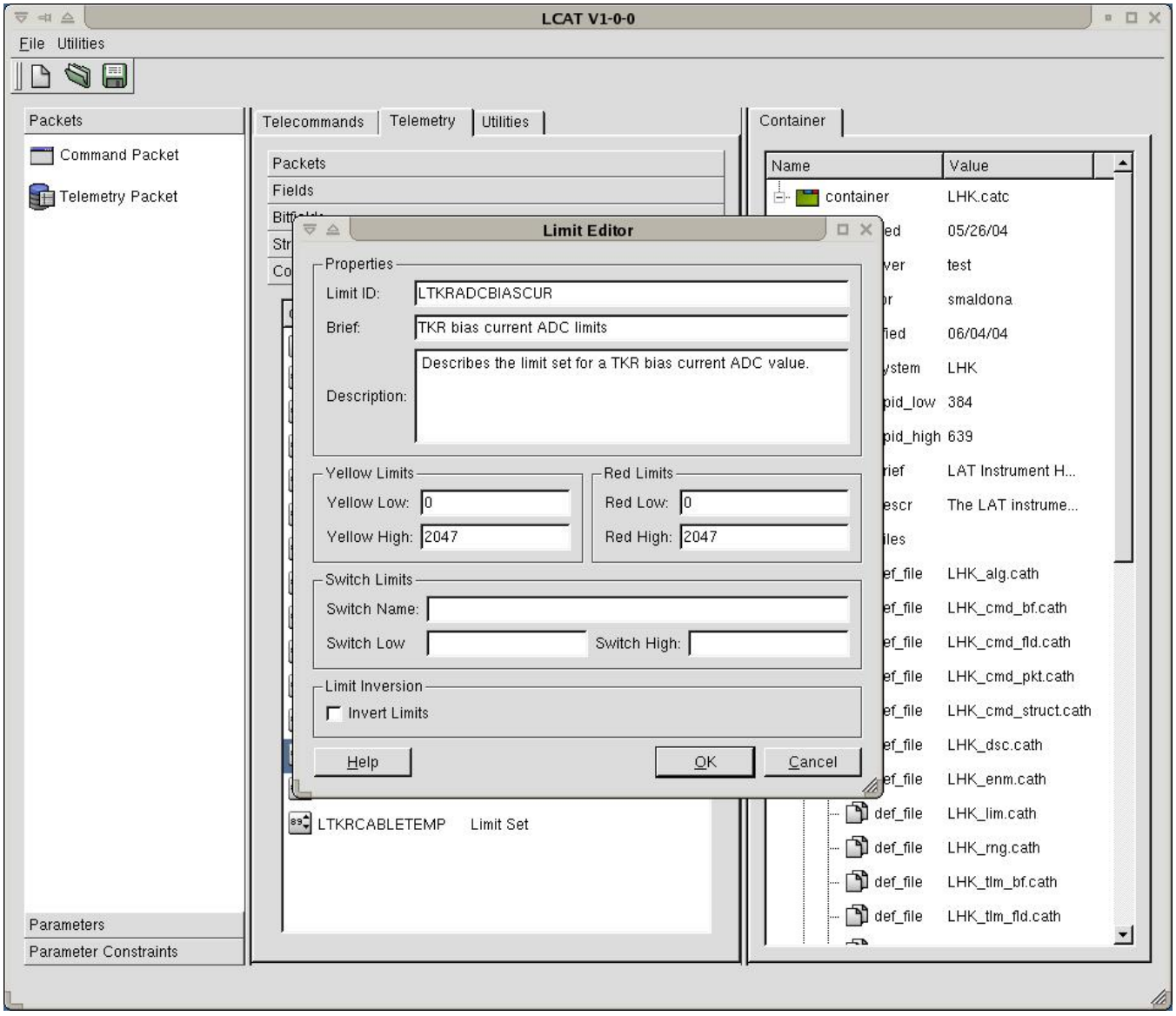


Figure 16 Limit Editor

2.5 Container Validation

Local validation checks for consistency within a container. Global validation checks locally and across all containers in the CMX tree.

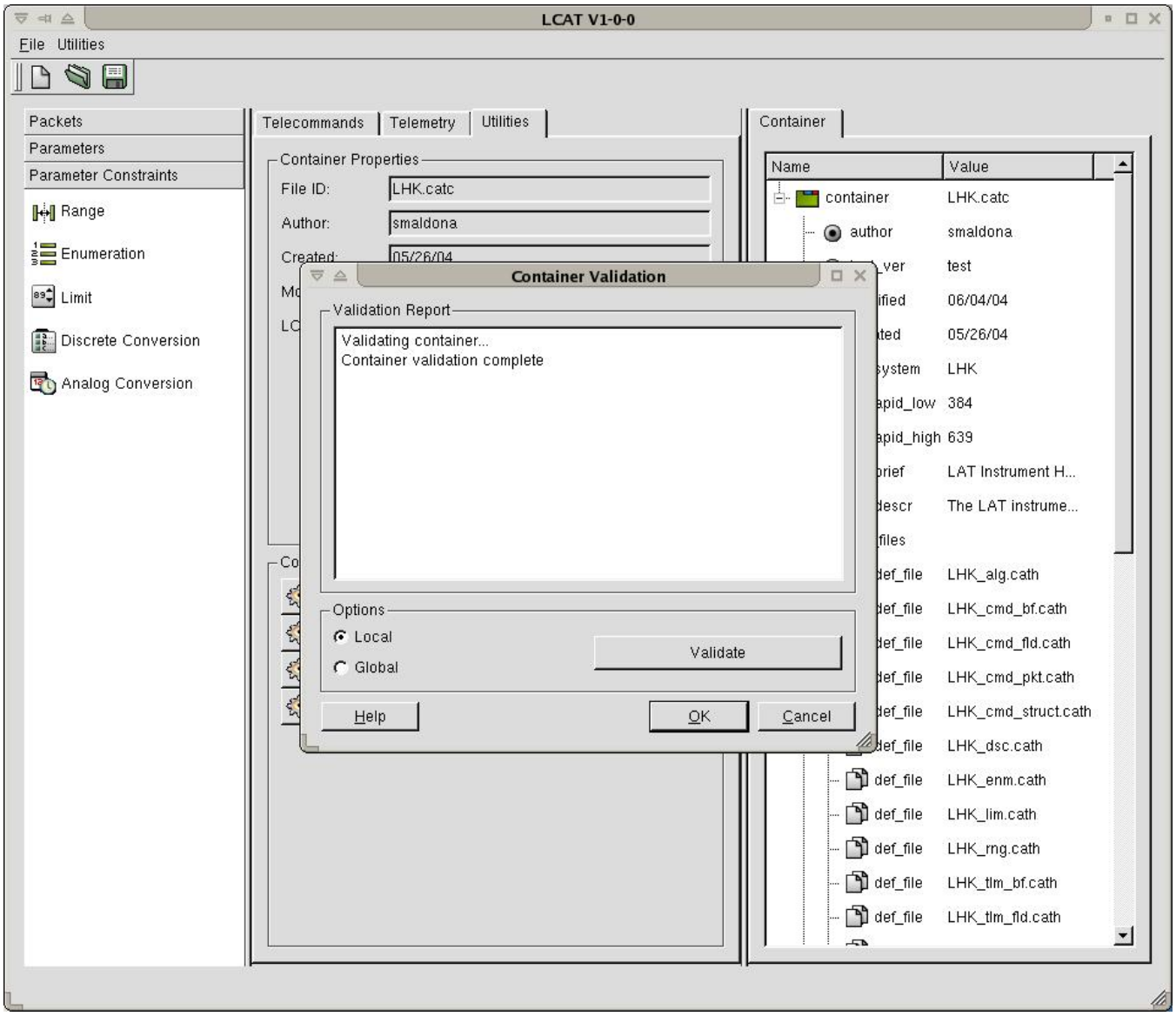


Figure 17 Container Validation

2.6 Adding ITOS Mnemonics

ITOS mnemonics must be specified for commands and telemetry packets.

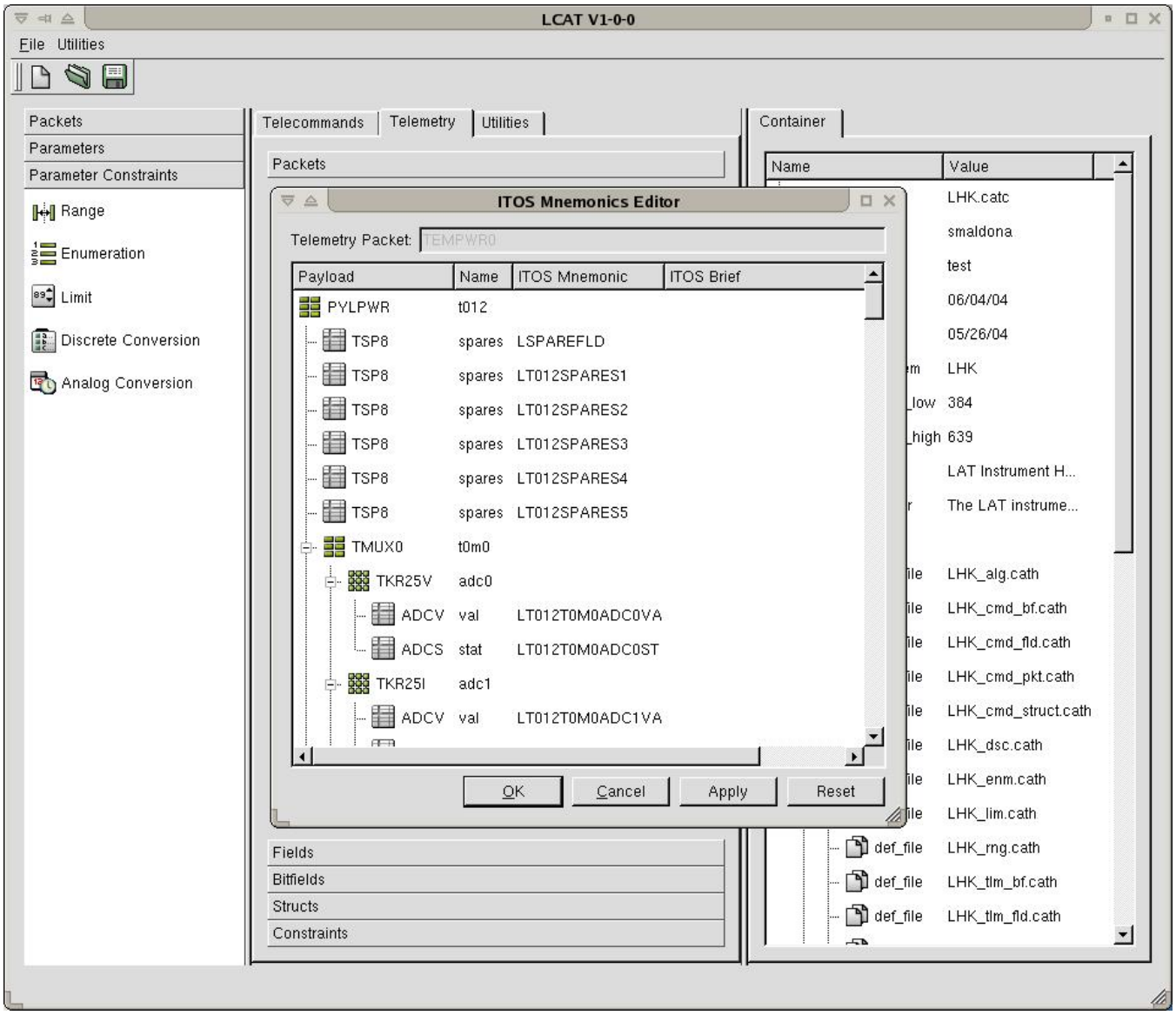


Figure 18 ITOS Mnemonic Editor

2.7 Generating ITOS

ITOS output files are generated in the /cat directory.

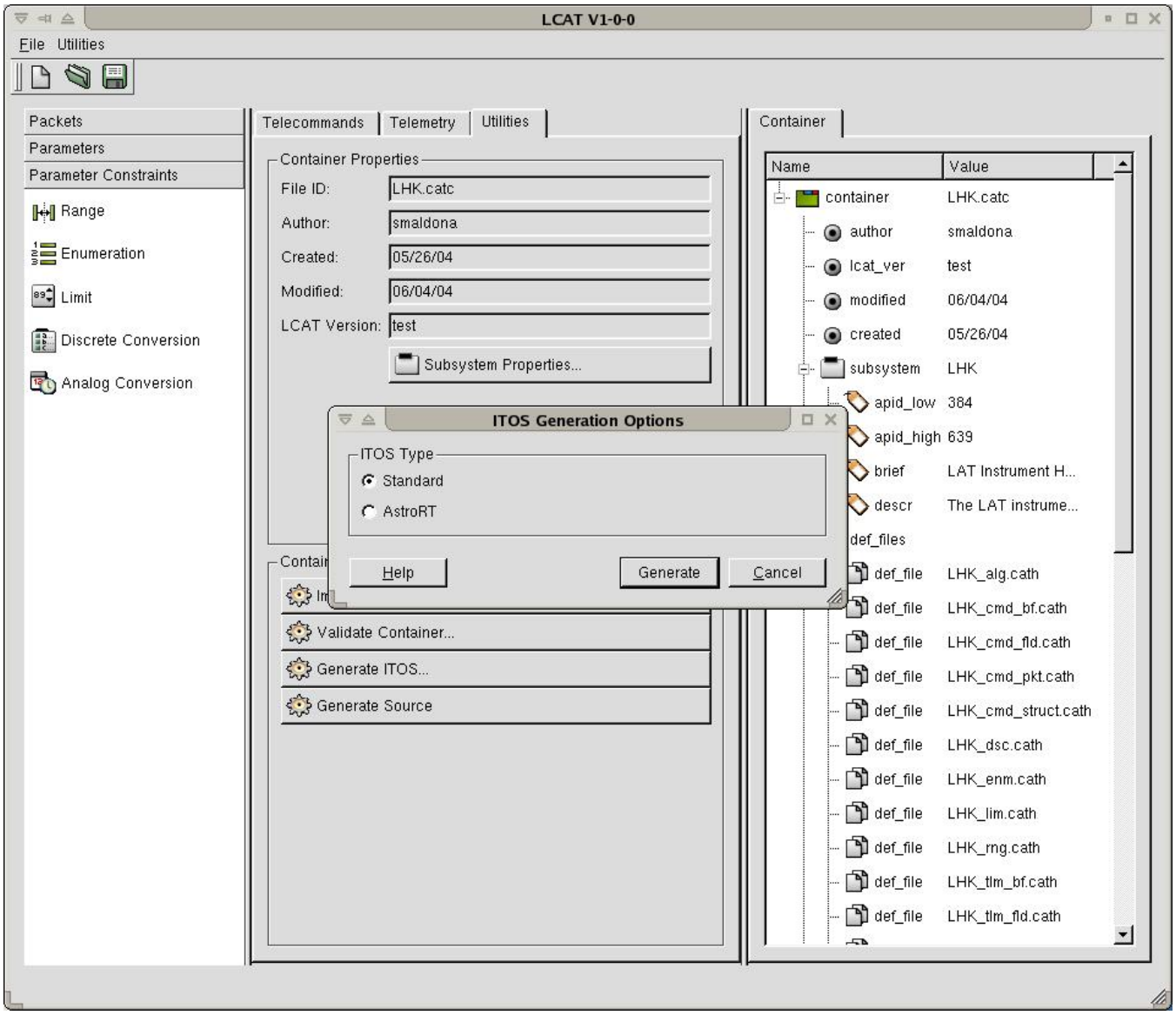


Figure 19 Generating ITOS

2.8 Generating Source Code

Source code outputs files are generated in the public header directory for the package, and in the /src directory.

2.9 LCAT Command Line Utilities

LCAT provides a command line interface for some application functionality. With an active CMX environment, the following LCAT commands are accessible:

2.9.0 lcat validate

Performs validation on the container and definitions files

```
lcat validate <container_file> [--local | --global]

 <container_file>  Filename of container to validate
 --global          Validate against all global packages
 --local           Validate local container
```

2.9.1 lcat generate

Generates output from a container and definitions files.

2.9.1.0 source

Generates source code output.

```
lcat generate source <container_file>
    [--cmd=<cmd_header_filename>]
    [--template=<cmd_src_template_filename>]
    [--tlm=<tlm_header_filename>]

<container_file> Filename of container
```

2.9.1.1 itos

Generates ITOS files

```
lcat generate itos <container_file>
    [--cmd=<itos_cmd_filename>]
    [--tlm=<itos_tlm_filename>]

<container_file> Filename of container
```

2.9.1.2 dependencies

Prints a dependency list for the container

```
lcat generate dependencies <container_file>

<container_file> Filename of container
```

2.9.1.3 xml

Prints an xml dump of the container including all referenced definition files.

```
lcat generate xml <container_file>

<container_file> Filename of container
```

3 LCAT Products

LCAT generates ITOS database files and C source code from validated container files.

3.0 ITOS

To generate an ITOS database, select “Generate ITOS” from the Utilities menu. Select the ITOS version in the subsequent dialog and press “Generate”. The generated ITOS files will be created in the cat directory.

3.0.0 ITOS Versions

Two type of ITOS output can be generated.

- Complete
    - Utilizes the “SSI” field to define the subsystem
- Limited (AstroRt)
    - No “SSI” records

3.1 C Source Code

LCAT generates C structs and function prototypes from a container file. To generate source code, select “Generate Source” from Utilities menu. The generated source files will be created in the public header directory and the src directory.

3.1.0 Packet Structs

Based on the container content, LCAT will generate a struct for each packet definition, parameter definition, and appropriate constraint.

3.1.1 ITC Routing Tables

The inter-task communication package (ITC) handles routing of telecommand packets to the appropriate FSW task. LCAT will generate the required ITC structs needed to register APIDs, function codes, and callback functions.

3.1.2 Callback Function Prototypes

A function prototype and callback function stub are generated for each command packet.

4 Installing LCAT

LCAT was developed using open source and GPL software freely available to the public. It was developed for the Sun UNIX and Linux platforms and is not compatible on any other operating system.

4.0 Prerequisites

4.0.0 Prerequisite Enumeration

LCAT employs a range of open source software products to handle XML files, graphical windows, and so on. The list prerequisite software is described below:

4.0.0.0 Full GUI LCAT

- Python 2.2.2 or later
- PyXML version 0.8.1 or later
- 4Suite version 1.0a3 or later
- Qt/X11 GPL/Non-Commercial version 3.1.1 or later – A platform independent C++ application development framework
- QScintilla version 1.53-x11-gpl or later – A port to Qt of the Scintilla C++ editor class
- SIP version x11-gpl-3.6 or later -- Python bindings for C++
- PyQt gpl version 3.6 or later – A python interface to Qt libraries
    - The path to the Qt /lib directory must be added to the installer's LD_LIBRARY_PATH environment variable before installing this module

4.0.1 Practical Experience With Prerequisites

Installing the LCAT prerequisite components is not without its frustrations. The following was generated by going through the installation process and noting any variances, special cases and so on. The installer in the following exercise had root privilege and could install to the canonical Unix areas. For sites where this is not practical, please omit the "make install" step (or its equivalent) and do what is acceptable at your site to make these products "visible":

- Python-2.2.2

  Build with default options: "./configure" → "make" → "make install"

- PyXML-0.8.3

  Build with default options: "python setup.py build" → "python setup.py install"

- 4Suite 1.0a3

  Build with default options: "python setup.py build" → "python setup.py install"

- Qt-x11-gpl-3.2.2
  To support Hippo draw, we need to configure Qt with –thread option: "./configure –thread", which will an extra multi-threaded library.

  But if we only want to run LCAT we can do without the multi-threaded option.

- QScintilla-1.54-x11-gpl-1.2

  Build with default options: follow instructions in README file.

- SIP-x11-gpl-3.8

  This package doesn't work straight out of the box on Sun. Apply the following patch to `siplib/sipQt.h`:

```
#include <Python.h>
#include <qobject.h>
#include <sip.h>
```

Figure 20 Section of code in `sipQt.h` before patch

```
#include <Python.h>
#if defined (truncate)
#  undef (truncate)
#endif
#include <qobject.h>
#include <sip.h>
```

Figure 21 Section of code in `sipQt.h` after patch

SIP can now we can build with default options: follow instructions in README file.

- PyQt-x11-gpl-3.8.1

  Build with default options: "build.py" → "make" → "make install"

4.1 Summary

Once all of the required software is installed, the user should have access to the LCAT command line and the GUI interface.