LAT Flight Software

LCAT User Manual

Type:	User Manual
Version:	V0-0-0
Author:	S.Maldonado
Created:	11 February 2004
Updated:	12 February 2004
Printed:	12 February 2004

Manual for the LAT FSW Telecommand and Telemetry tool.

Contents

Figures

Tables

Error! No table of figures entries found.

0 Introduction

0.0 Overview

The LCAT (**L**AT **C**ommand **A**nd **T**elemetry) 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 used 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 an XML file. 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 "resource" files. A single top level "container" file holds a reference to each resource file and allows for the sharing or reuse of data content between them.

All LCAT specific definition files use the file extension ".lct".

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 (resources), and lists of command and telemetry mnemonics 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 APID range for which all its associated telecommands and telemetry are valid.

1.1.1 Command References

Containers hold references to existing command definitions which reside in separate resource files.

1.1.2 Telemetry References

TBD.

1.2 Resource Files

Resource files contain the actual definitions of telecommands, telemetry, fields, or constraints. These definitions can be reused throughout the subsystem. Generally, a separate resource file should be created to hold commands, fields, and constraints.

1.2.0 Command Definitions

A telecommand is identified by an APID and function code combination, and is referenced by a unique mnemonic. All command definitions must be unique within the entire LAT command set.

1.2.0.0 Command Mnemonic

The mnemonic is the name of the command and can be up to 16 characters long.

1.2.0.1 APID

Application ID

1.2.0.2 Function Code

Command type indicator

1.2.0.3 Length

Command length in bits

1.2.0.4 Payload

The command payload consists of a set of references to field definitions, and associated constraints on the field values. Each payload item also describes the starting byte of the field within the packet, calculated from the start of the secondary packet header.

1.2.1 Field Definitions

Fields describe the command packet's data contents.

1.2.1.0 Field Name

A descriptive name for the field. These are unique only within a command.

1.2.1.1 Data Type

Indicates the type of the value for the field (int, float, string, etc)

1.2.1.2 Length

Field length in bits.

1.2.2 Constraint Definitions

A constraint describes how a field value is interpreted or limited.

1.2.2.0 Range

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.

1.2.2.1 Enumerations

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.

1.2.2.1.1 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.

1.2.3 Telemetry Definitions

TBD

1.3 Organizational Hierarchy

- container - .lct
    - subsystem
        - apid low
        - apid high
        - brief
        - description
    - resources
        - resource name
    - command references
        - command name
- resource - .lct
    - command
        - apid
        - function code
        - length
        - brief
        - description
        - payload

        - field name
        - constraint name
        - start byte
    - field
        - data type
        - length
        - brief
        - description
    - range
        - low limit
        - high limit
        - brief
        - description
    - enumeration
        - brief
        - description
        - member
            - value
            - brief
            - description

2 The LCAT Application

The LCAT application provides a main workspace, drag and drop icons, 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 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.

2.0.0 The Cat Directory

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

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

2.0.1 Launch LCAT

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

2.0.2 The Main Window

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

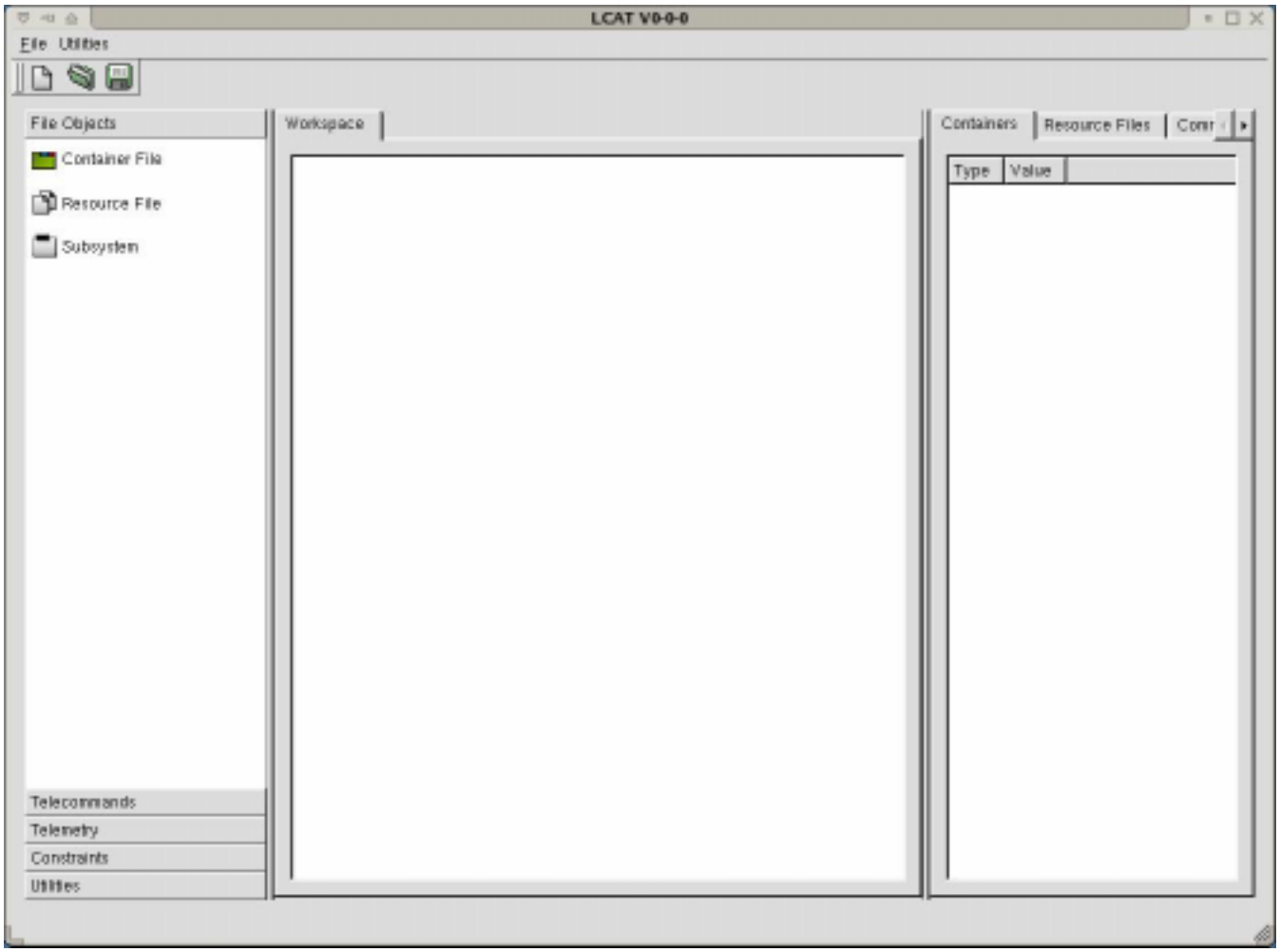

Figure 1 LCAT Main Window

2.1 Containers

2.1.0 Creating a New Container

Create a new container file by selecting “File Objects” toolbox, and clicking the “Container File” button, or by selecting File->New from the toolbar. Enter the path and filename of the new container in the dialog box.

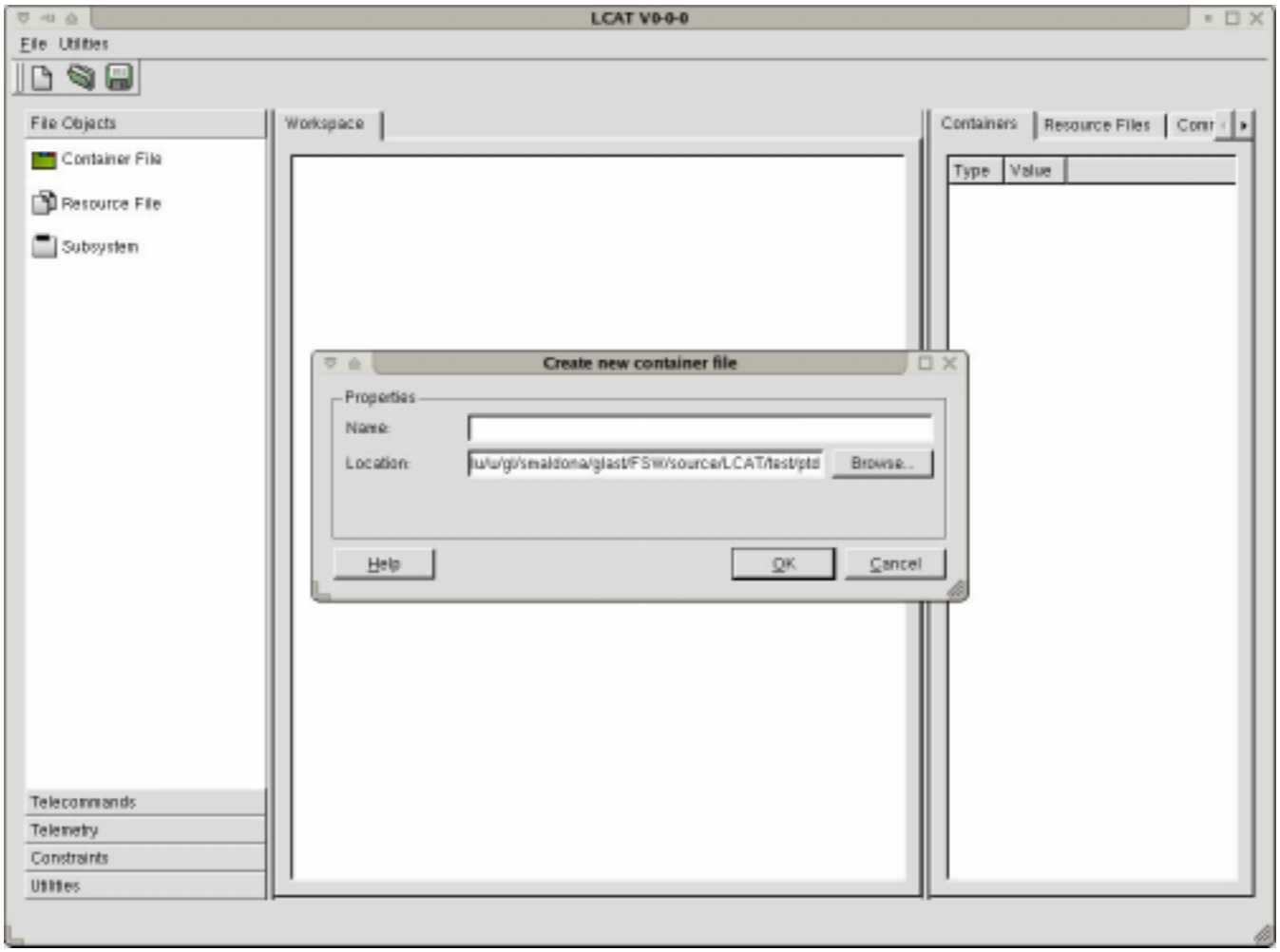

Figure 2 New Container Dialog

2.1.1 Editing Container Contents

Once a new, container is created, an icon will appear in the workspace. Double-click the container icon to view its contents. An empty subsystem object is created for you by default. To remove an item from a container, select the item, right-click to display a context menu, and select “Remove”. The reference to the item will be removed.

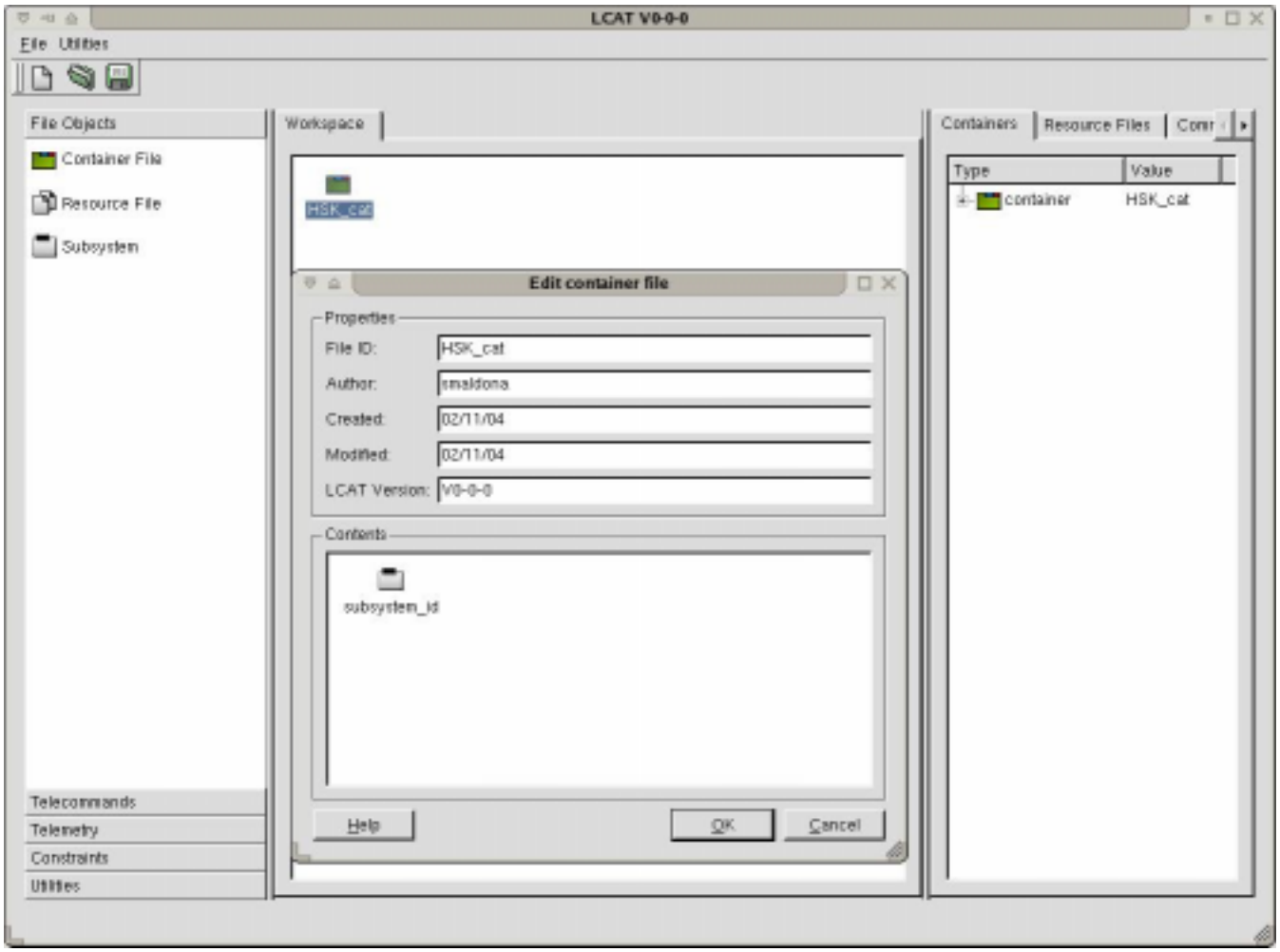

Figure 3 Container Editor Dialog

2.1.2 Editing the Subsystem Object

Double the subsystem icon 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.

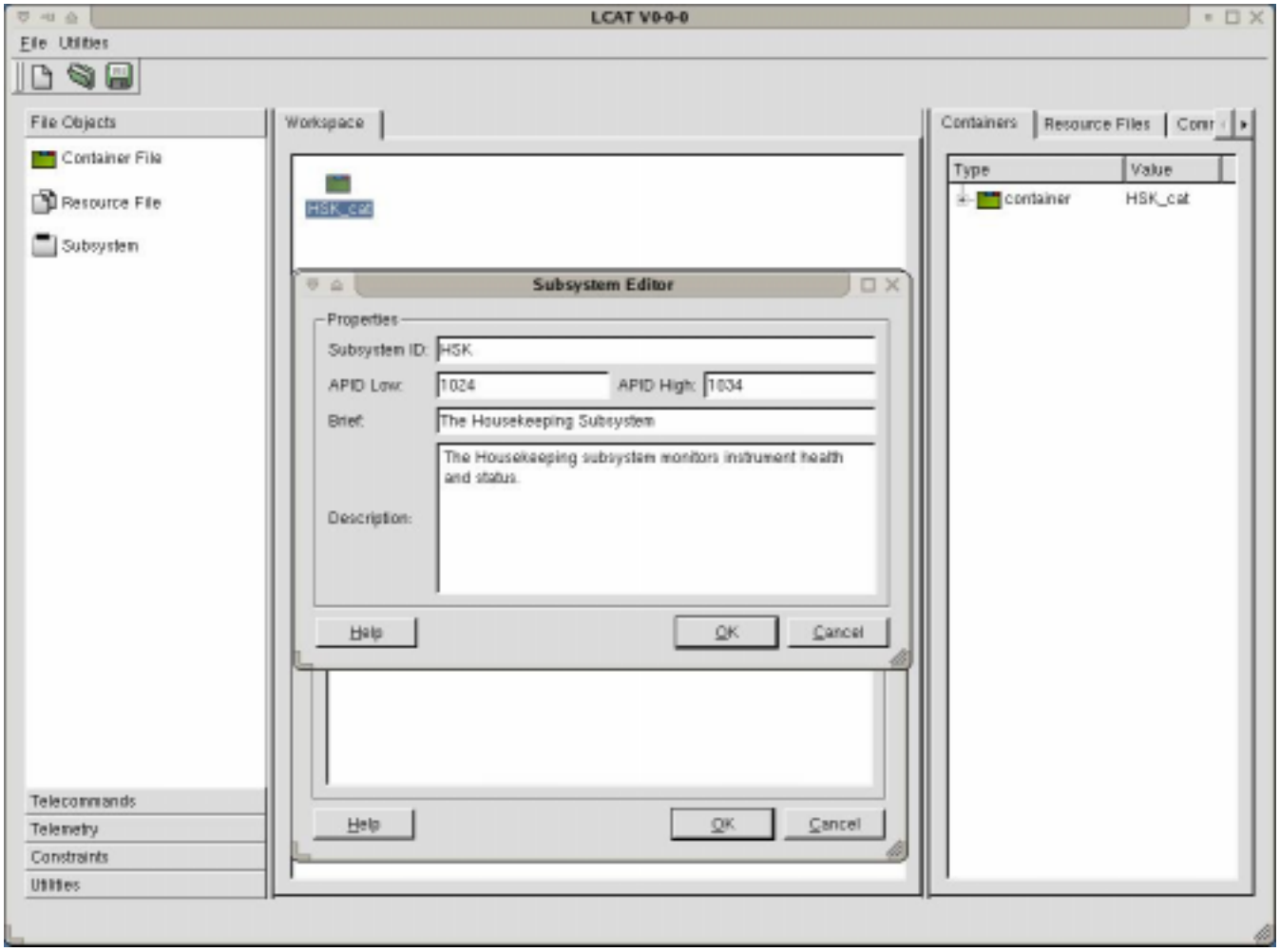

Figure 4 Subsystem Editor Dialog

2.2 Resource Files

Each resource file holds all definitions for commands, fields, or constraints. Create separate resources file for commands, fields, ranges, and enumerations by clicking the appropriate button in the toolbox. An icon will appear in the workspace for each new resource created.

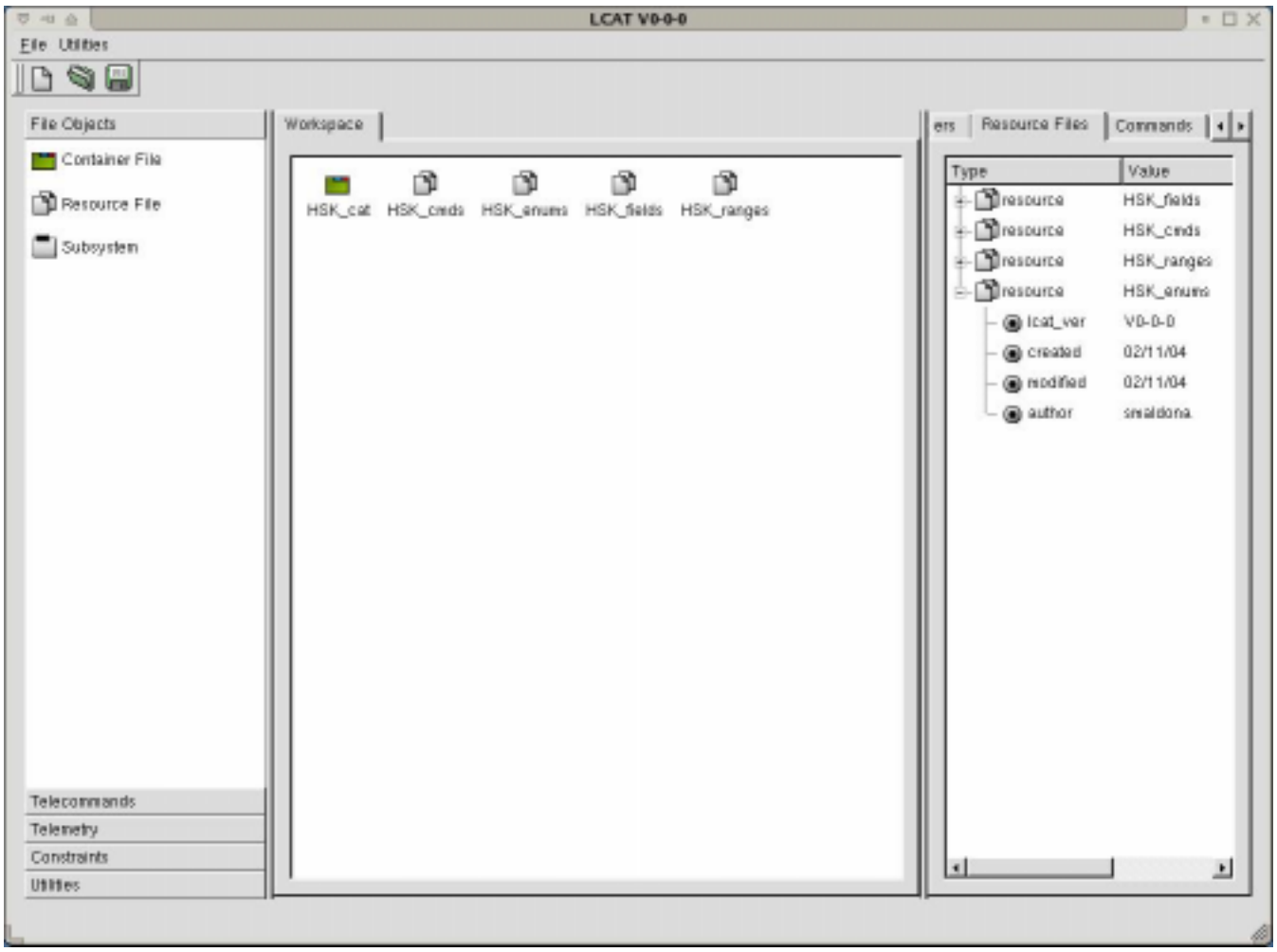

Figure 5 LCAT Resource Files

2.3 Telecommands

Now that the container and resource files are created, new commands, fields, and constraints can be added to them.

2.3.0 Create a New Telecommand

To create a new telecommand, select Telecommands in the toolbox and click the command button. Enter command data into the editor. Command length will be calculated for you, based on the payload. Press OK to accept.

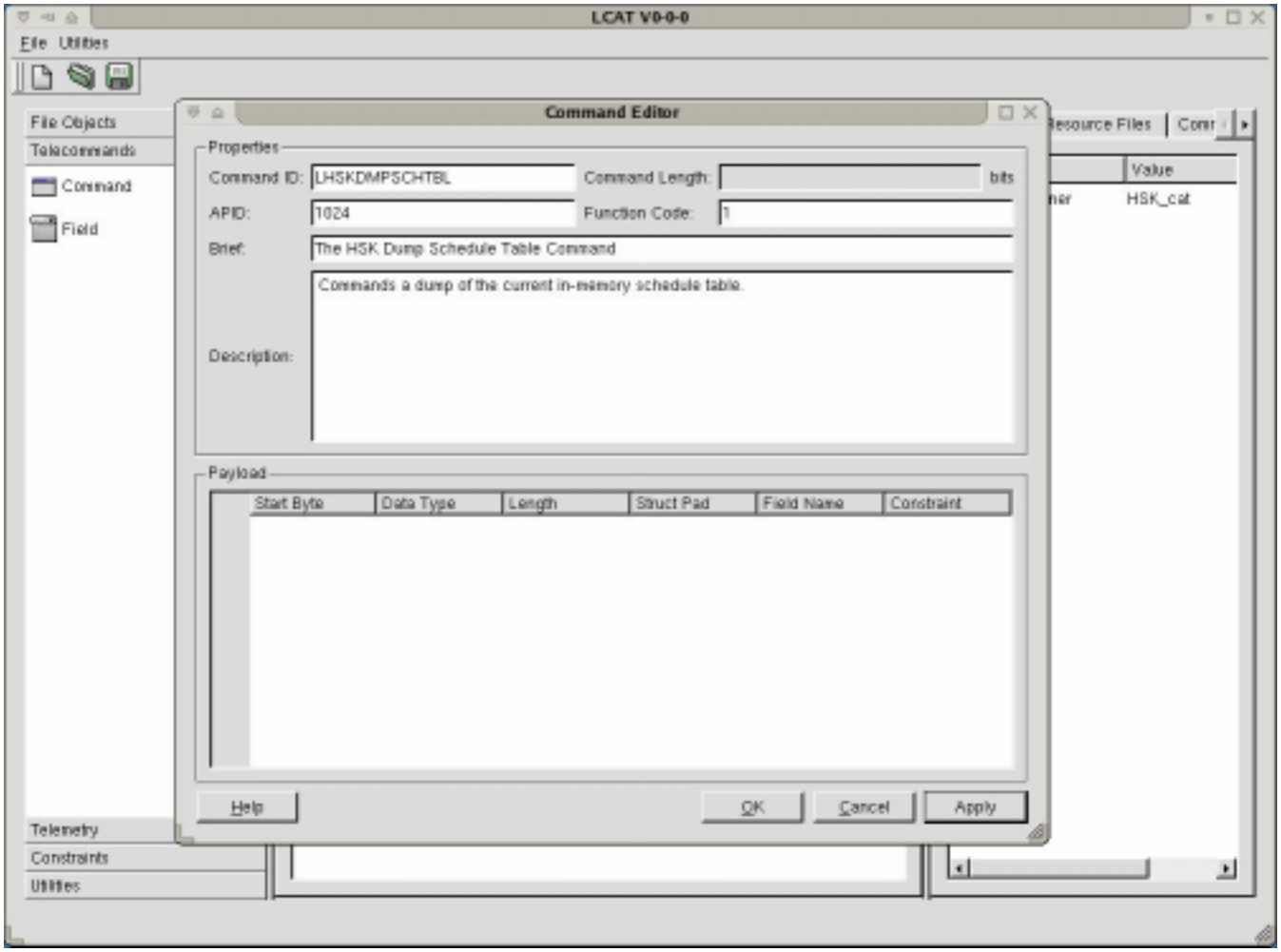

Figure 6 Command Editor Dialog

A new command icon has been created in the workspace. Double-click the icon to edit the command properties.

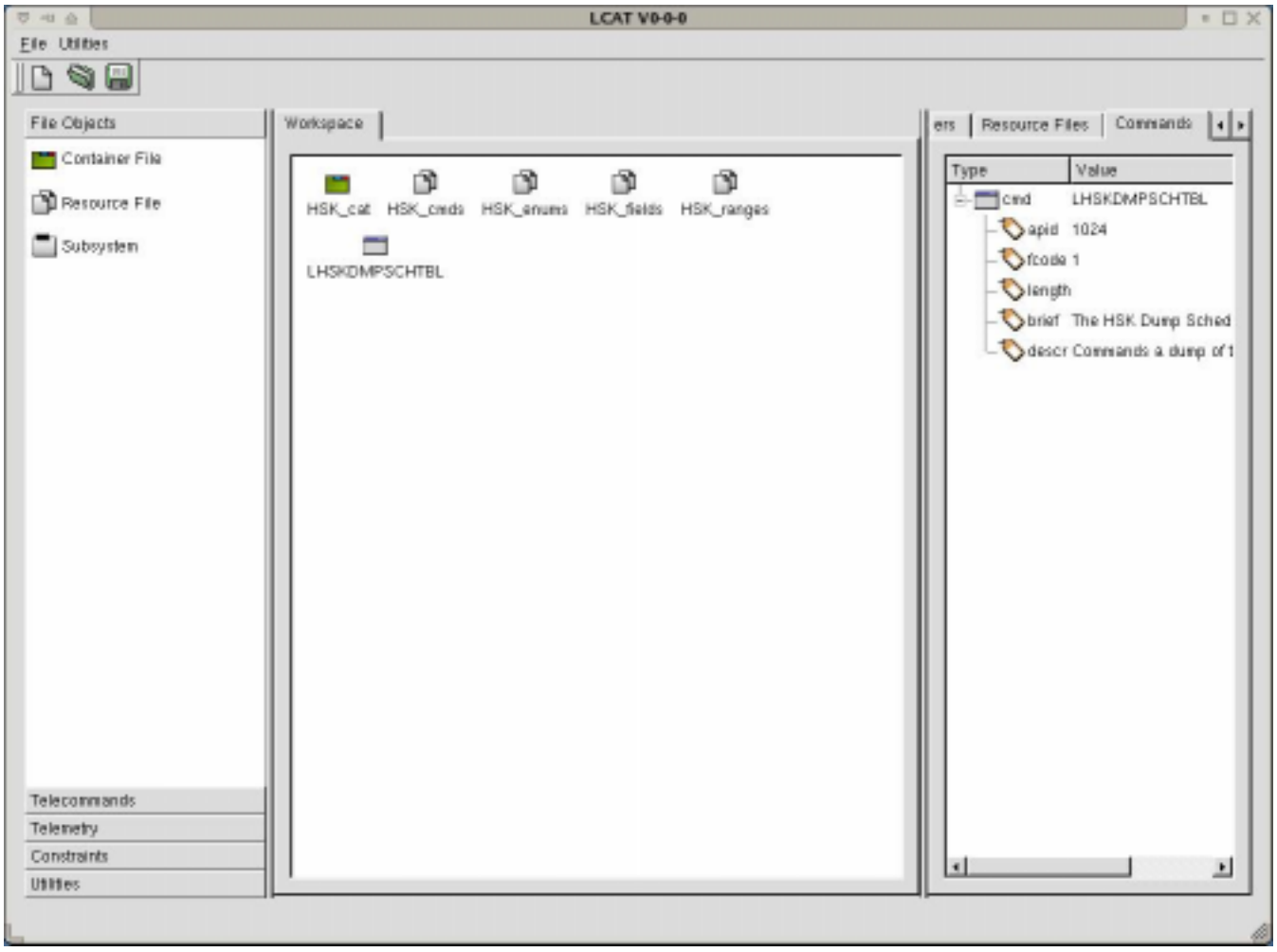

Figure 7 Command Icons

2.3.1 Adding Fields

Create a new field in the same manner as a command. A new field icon will be placed in the workspace. Drag the field icon into the command to add the field. Repeat for each new field definition. The command tab view will display the added fields as payload item. Create any constraint objects at this time.

2.3.2 Adding Constraints

Constraints are created in the same manner as commands. A new constraint icon will appear in the workspace. You cannot drag a constraint into a field. Payload items specify the constraint on a field.

2.3.2.0 Ranges

Create a new range constraint in the same manner as a command. A new range icon will appear in the workspace. No need to drag a

2.3.2.1 Enumerations

When creating enumeration, member values must also be created. Drag the member icons into the enumeration icon to add a member.

2.3.3 Editing the Command Payload

To edit or view the payload, double-click the command icon to bring up the editor. To change the position of a payload item or remove it, select it in the payload table and right-click to display a menu. To specify a constraint on the field, select it from the constraint column. All existing constraint definitions will appear as valid selections.

2.3.3.0 The Command Payload Table

The command payload table displays the current fields container in the command. The start byte value counts from the start of the packet header. The primary packet header is 4 bytes, and the secondary header is 2 bytes, so payload always starts at byte 8. The struct pad column display how many pad bytes will have to be added after the item, when generating the associated C struct. Try to order fields to minimize padding.

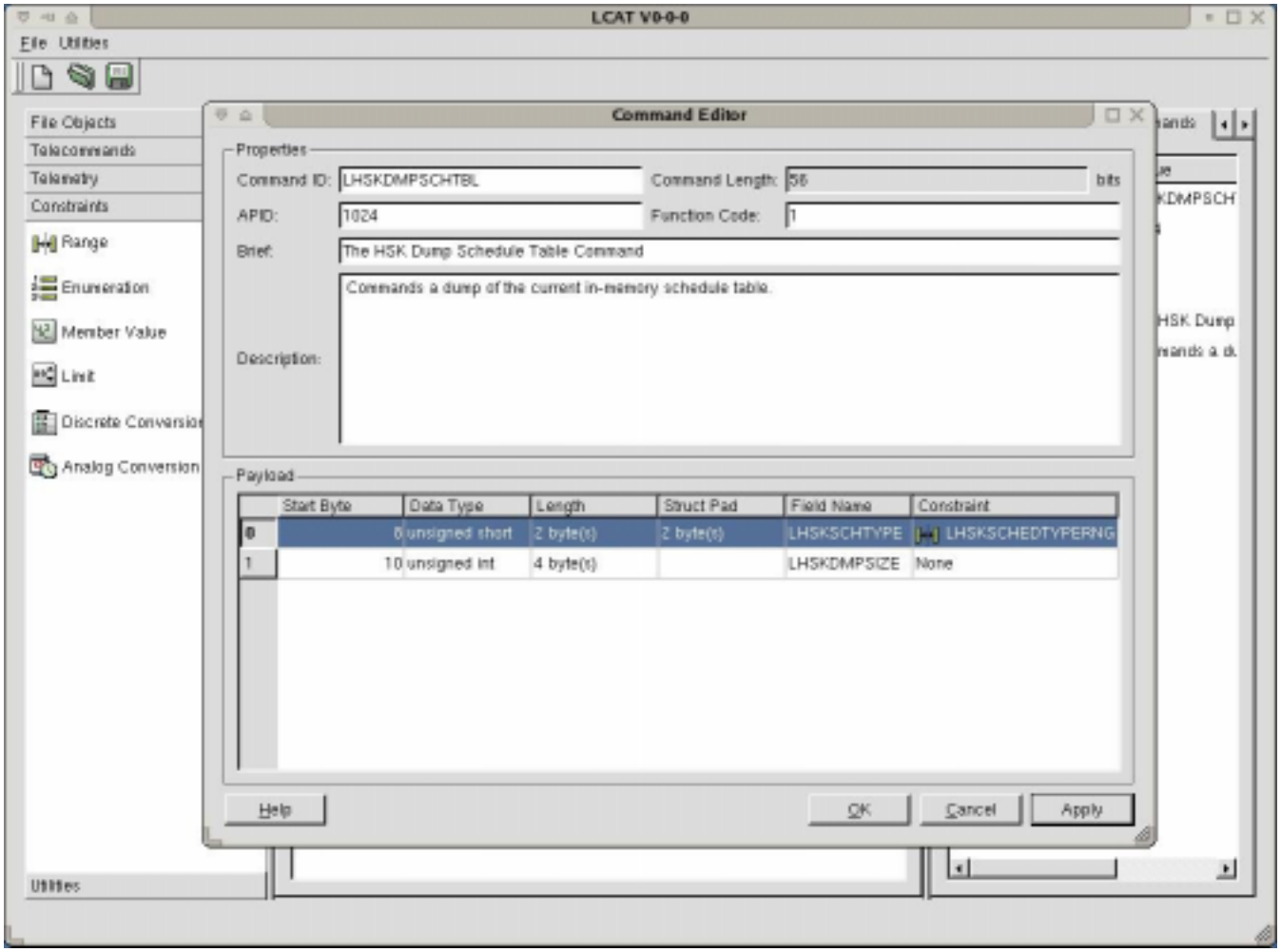

Figure 8 Command Payload Editing

2.4 Telemetry

TBD

2.5 Populating Container and Resource Files

Once the definition objects have been defined, they must placed in the appropriate storage object.

2.5.0 Adding Commands to a Container

To add a command to a container, drag the command icon onto the container. The container tab view will display a new command reference, and the command item will stay in the workspace. To edit or remove the command, double-click the container icon, select the command reference, and right-click to show the context menu.

2.5.1 Adding Definition Objects to Resource Files

Once the commands have been added to a container, they need to be placed in their appropriate files. Drag each object into its associated resource file. Finally, save each resource file.

2.5.2 Adding Resource Files to a Container

Once all of the resource files are complete, drag each one into the container. A resource reference will be placed in the container and will be displayed in the container tab view.

2.6 Reusing Resource File Contents

To reuse a object from a resource file, open the resource file editor, select the item, right-click to show the context menu, and select “Return to Workspace”. The selected item will appear in the workspace, and can be reused or edited as necessary. Remember to return the item to it’s resource file when complete.

3 LCAT Products

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

3.0 ITOS

To generate an ITOS database, highlight a container and select “Generate ITOS” from the toolbox or 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
    - Allow for bitfields in command payload specification (TBD)
- Limited (AstroRt)
    - No “SSI” records
    - No command bitfields

3.1 C Source Code

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

3.1.0 ITC Function Tables

TBD

3.1.1 Command Payload Structs

A struct is created for each command’s payload

3.1.2 Callback Function Prototypes

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

4 Installing LCAT

TBD