Core Flight System
Command and Data Dictionary Utility
User's Guide

Engineering Directorate
Software, Robotics, and Simulation Division

Version 1.4.14
August 2018

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas 77058-3696

Contents

Figures

Tables

1.0 Description

The Core Flight System (CFS) Command and Data Dictionary (CDD) utility, or CCDD, is a software tool for managing the command and telemetry data for CFS and CFS applications. CCDD is written in Java™ and interacts with a PostgreSQL database, so it can be used on any operating system that supports the Java Runtime Environment (JRE) and PostgreSQL. CCDD is released as open source software under the NASA Open Source Software Agreement, version 1.3, and is hosted on GitHub.

The CCDD application uses tables, similar to a spreadsheet, to display and allow manipulation of telemetry data structures, command information, and other data pertinent to a CFS project. The data is stored in a PostgreSQL database for manipulation and data security. The PostgreSQL database server can be run locally or centralized on a remote host for easier access by multiple users. Data can be imported into the application from files in comma-separated values (CSV), JavaScript Object Notation (JSON), electronic data sheet (EDS), and extensible markup language (XML) telemetric and command exchange (XTCE) formats. Data can be exported from the application to files in CSV, JSON, EDS, and XTCE formats. The CCDD tables also allow simple cut and paste operations from the host operating system's clipboard. To make use of the project's data, CCDD can interact with Java Virtual Machine (JVM)-based scripting languages via a set of supplied data access methods. Using scripts, the user can translate the data stored in the CCDD's database into output files. Example scripts for creating common CFS related output files are provided in four of these scripting languages. An embedded web server can be activated, allowing web-based application access to the data. Figure 1 shows the basic relation between CCDD and external sources.

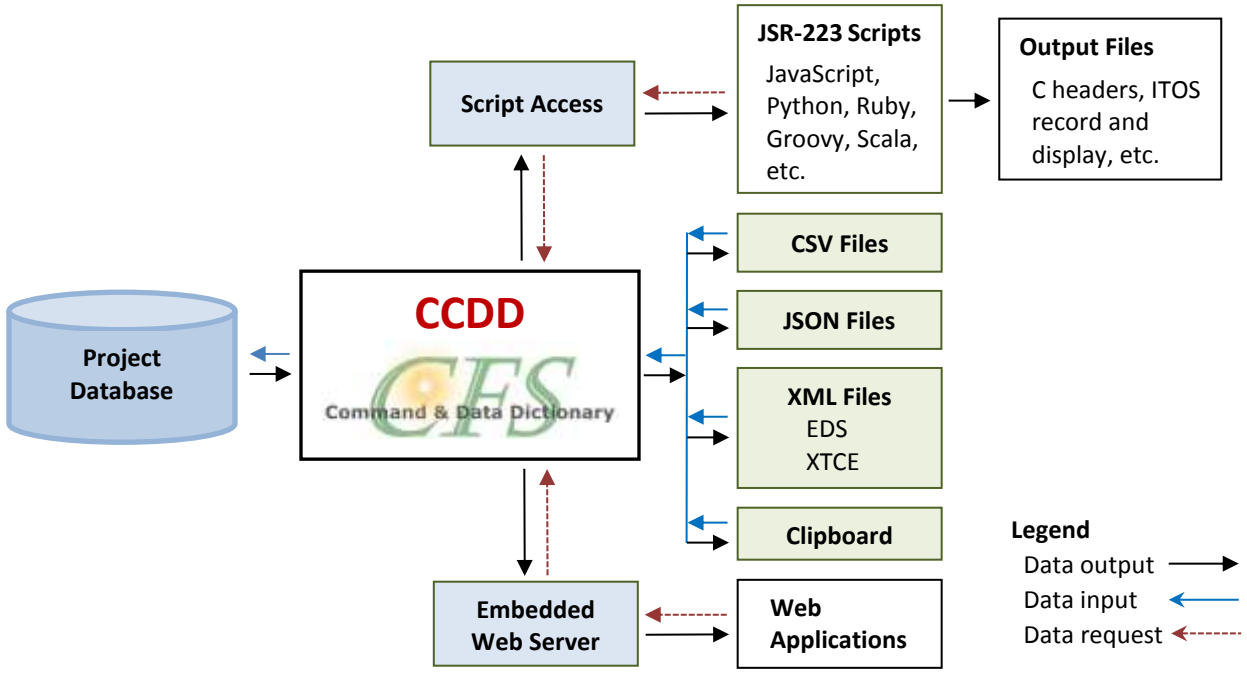


Figure 1. CCDD inputs and outputs

Questions or comments concerning this document or the CCDD application should be addressed to:

Johnson Space Center
Software, Robotics, and Simulation Division
Spacecraft Software Engineering Branch, Mail Code ER6
Houston, TX 77058

2.0 Requirements

CCDD is written based on the following Java and PostgreSQL versions:

- JavaSE 1.7
  - Developed in Linux using JavaSE 1.7 and tested on CentOS 6 Linux, Microsoft Windows 7, and Apple OS X using JavaSE 1.8
- PostgreSQL 8.4

CCDD comes with the following Java Database Connectivity (JDBC) driver, embedded Jetty web server, and JavaScript Object Notation (JSON) versions:

- JDBC driver 9.4-1207 (type 4)
- Jetty 9.2.18.v20160721
- JSON simple 1.1.1

Scripting language testing was performed using the following languages and versions:

- JavaScript (ECMAScript) 1.8 (Mozilla Rhino 1.7 release 3)
- JavaScript ECMA – 262 Edition 5.1 (Oracle Nashorn 1.8.0_131)
- Python 2.7 (Jython 2.7)
- Ruby 9.0.1.0 (JRuby 9.0.1.0)
- Groovy 2.4.4 (Groovy Scripting Engine 2.0)
- Scala 2.12.4 (Scala REPL 2.0) - *requires Java 1.8*

EDS and XTCE XML schema versions:

- EDS 1.3.112.4.17.1
- XTCE 1.1

Compatibility with other versions, in particular earlier ones, is not guaranteed.

3.0 Installation

To install CCDD copy the Java archive (jar) file `CCDD.jar`, the user's guide file `CCDD_Users_Guide.pdf`, and the library folder `CCDD_lib` and its contents to a folder. The application requires read/write access to a folder so that event log file(s) can be created; the default folder for the log files is the file from which the application is executed though this can be changed (see paragraph 4.2.5 for further information on event logs and paragraph 4.1 for changing the log file path). **Java and PostgreSQL must be installed before the application can be used.**

- To install Java, go to www.java.com and locate the installation instructions appropriate for the operating system on which the application is to be run.
- The PostgreSQL relational database management system is available for download from www.postgresql.org. The format appropriate for the target operating system must be used. Once installed, PostgreSQL must be configured prior to use by the application. Configuration includes setting up the PostgreSQL server as a background service, creating database users and roles within the PostgreSQL server, and setting the desired level of password authentication. Extensive information on configuring PostgreSQL is available from www.postgresql.org.

The `CCDD_lib` folder contains versions of the library files required by CCDD for PostgreSQL interaction, the embedded web server, and XML import/export format conversions. Updated versions of these

libraries can be used in place of those in this folder as described in paragraph 4.1. Information on these libraries can be found at the following:

- The PostgreSQL JDBC driver is located at jdbc.postgresql.org. *Note that version of Java installed determines which JDBC driver to use*.
- Jetty must be installed in order to use the web server. Jetty is located at http://www.eclipse.org/jetty/. *Note that version of Java installed determines which Jetty version to use*.
- The JSON conversion library is located at https://code.google.com/archive/p/json-simple.
- Information regarding EDS is located at http://www.ccsds.org.
- Information regarding XTCE is located at http://www.omg.org/space/xtce.

CCDD supports the use of JVM-based scripting languages. At least one of these languages must be installed for the application to make use of CCDD's project-data-to-script-language interface. Only the scripting language(s) intended for use with the application need to be installed. The application was tested with five of the available languages: JavaScript, Python, Ruby, Groovy, and Scala. Details are provided in this in subsequent sections of this guide on the use of these five scripting languages; installation and use of other languages should be similar. The CCDD package provides examples of common scripts in the JavaScript, Python, Ruby, and Groovy languages.

The scripting languages are not part of the CCDD package and must be installed separately on the platform from which the CCDD application is launched. The following links can be used to find further information on downloading and installing the scripting languages.

- *JavaScript®* is part of the JRE download and installation from www.java.com, so no further installation is necessary to use this scripting language. More information on JavaScript can be found at developer.mozilla.org.
- *Python™* scripting is implemented using *Jython*, the Python implementation for Java. Jython can be downloaded from www.jython.org.
- *Ruby* scripting is implemented using *JRuby*, which implements Ruby in Java. JRuby is available for download from jruby.org.
- *Groovy* can be downloaded from www.groovy-lang.org.
- *Scala* can be downloaded from www.scala-lang.org.

4.0 Operation

4.1 Getting Started

To run the application open a command prompt window and type:

```
java –classpath class_paths CCDD.CcddMain [args…]
```

where `class_paths` includes the paths and .jar file names for the CCDD application (`CCDD.jar`) and supporting libraries, separated by colons (:) with no intervening spaces, and `args` are optional command line arguments in the form:

```
[<- or />command [value] […]]
```

Versions of the JDBC driver, Jetty server, and JSON conversion libraries are included in the `CCDD_lib` folder. If the default version is incompatible with the installation or an updated version is desired then the defaults can be overridden by including the library path(s) and .jar file name(s) in the `class_paths` argument (or by overwriting the file(s) in the `CCDD_lib` folder). The library files needed are shown below; <*version*> is the specific version number of the installed file that is part of the file's name.

```
<JDBC path>/postgresql-<version>.jar
<Jetty path>/lib/jetty-http-<version>.jar
<Jetty path>/lib/jetty-io-<version>.jar
<Jetty path>/lib/jetty-security-<version>.jar
<Jetty path>/lib/jetty-server-<version>.jar
<Jetty path>/lib/jetty-servlet-<version>.jar
<Jetty path>/lib/jetty-util-<version>.jar
<Jetty path>/lib/servlet-api-<version>.jar
<JSON path>/json-simple-<version>.jar
```

The library class path(s) must be specified for each scripting language other than JavaScript. The library files required by Ruby, Python, Groovy, and Scala are shown below; <*version*> is the specific version number of the installed file that is part of the file's name. Library files for other scripting languages should follow a similar format.

```
<JRuby path>/lib/jruby.jar
```

Note: the JRuby reference must precede the other scripting language library references

```
<Jython path>jython.jar
<Groovy path>/lib/groovy-<version>.jar:<Groovy path>/lib/groovy-jsr223-
<version>.jar
<Scala path>/lib/scala-compiler.jar
```

Note: If running CCDD in Java 9 or 10 the startup command must be amended by adding the option to include the module containing the JAXB library:

```
java --add-modules java.xml.bind –classpath class_paths CCDD.CcddMain [args…]
```

Each command line argument consists of a command, preceded by either a '-' or '/', followed by a space, then the command value. The available commands and acceptable values are described in Table 1. The commands can be entered in any order. If the same command is entered more than once then each instance is applied/executed in the order of appearance on the command line. The commands are not case-sensitive, so "-user" is the same as "-USER", "-User", etc.

The `import` and `export` commands require one or more sub-commands. The sub-commands are listed in Table 1 directly beneath the entries for the export and import commands. The sub-commands have the same constraints as the commands, as described in the previous paragraph. However, the entire list of sub-commands must be bounded by single or double quotes. For example, the arguments for an export command would look similar to:

```
export "-tablePaths tableName -filePath /user/name/exportFolder -shutdown"
```

Single quotes may be used to bound command values within the export sub-command string if the string is bounded by double quotes, and vice versa.

Command		Description	Value	Default Value
backup		Sets the file path and name to which to automatically backup the project database once it is successfully connected. The extension ".dbu" is automatically appended to the file name if not already present. Only applies to the first successful connection	File path and name of the project backup file	*None*
command		Selects whether or not to display event log command messages	"true" to display event log command messages in the main application window; "false" to hide event log command messages. The value text is case insensitive	true
events		Selects whether or not to display all event log messages	"true" to display all event log messages in the main application window; "false" to hide all event log messages. The value text is case insensitive	true
export		Exports the specified table(s) to the specified file	Quoted (single or double) string containing one or more export sub-commands. See paragraph 4.9.3.8 for details	*None*
export sub-commands	classification1	(XTCE only) Sets the first level classification used in the XML headers. See paragraph 4.9.3.8.4 for details	Text	DOMAIN
	classification2	(XTCE only) Sets the second level classification used in the XML headers. See paragraph 4.9.3.8.4 for details	Text	SYSTEM
	classification3	(XTCE only) Sets the third level classification used in the XML headers. See paragraph 4.9.3.8.4 for details	Text	INTERFACE
	endianess	(EDS, XTCE) See paragraph 4.9.3.8 for the description of the **Endianess Big** and **Little** radio buttons	Sets the appropriate XML tags to indicate the endianness of the table data; "big" for big endian, "little" for little endian. The value text is case insensitive	big

Command		Description	Value	Default Value
export sub-commands	externalFileName	(EDS, XTCE) See paragraph 4.9.3.8.4 for the description of the **Enter script file name** input field	Name of the script, including its folder path, containing the export methods that replace the internal EDS or XTCE export methods. Omit this command to use the internal methods	*None*
	filePath	See paragraph 4.9.3.8 for details. The file path is required by the `export` command when exporting in the EDS or XTCE formats, or in the CSV or JSON formats when `singleFile` is true	Path for the folder into which the export file is placed	*None*
	format	See paragraph 4.9.3.8 for details	Determines the output file format: "csv" for CSV, "eds" for EDS, "json" for JSON, or "xtce" for XTCE. The value text is case insensitive	csv
	includeProjectFields	See paragraph 4.9.3.8 for the description of the **Include project data fields** check box	"true" or "false". The value text is case insensitive	false
	includeReservedMsgIDs	(CSV, JSON) See paragraph 4.9.3.8 for the description of the **Include reserved message IDs** check box	"true" or "false". The value text is case insensitive	false
	includeVariablePaths	(CSV, JSON) See paragraph 4.9.3.8 for the description of the **Include variable paths** check box	"true" or "false". The value text is case insensitive	false
	isHeaderBigEndian	(EDS, XTCE) See paragraph 4.9.3.8 for the description of the **Headers are big endian** check box	"true" or "false". The value text is case insensitive	true
	overwriteFile	See paragraph 4.9.3.8 for the description of the **Overwrite existing file(s)** check box	"true" or "false". The value text is case insensitive	false
	replaceMacros	See paragraph 4.9.3.8 for the description of the **Substitute macro values for macro names** check box	"true" or "false". The value text is case insensitive	false

Command		Description	Value	Default Value
	singleFile	(CSV, JSON) See paragraph 4.9.3.8 for the description of the **Store in Multiple files** and **Single file** radio buttons	"true" or "false". The value text is case insensitive	false
export sub-commands	tablePaths	Sets the paths for one or more tables or table groups to export. At least one table or group is required by the export command. See paragraph 4.9.3.8 for details	Names of the table(s) and/or group(s) to export. Group names must be preceded by 'Group:'. If multiple tables/groups are specified the table/group names must be separated by a plus (+)	*None*
	validationStatus	(EDS, XTCE) Sets validation status used in the XML headers. See paragraphs 4.9.3.8.2 and 4.9.3.8.4 for details	Text	Working
	version	(EDS, XTCE) Sets the version used in the XML headers. See paragraphs 4.9.3.8.2 and 4.9.3.8.4 for details	Text	1.0

Command	Description	Value	Default Value
execute	Runs the supplied script association(s), or script(s) using the supplied table(s) and/or group(s). The application's graphical user interface (GUI) is not displayed; exits upon completion of the script(s). See paragraph 4.9.5.1 for more detail	Script association name or script file name and associated table(s). Script file paths are required if the script is in a folder other than the one from which the application is executed. If the script requires one or more tables to be specified then the table name(s) and/or group name(s) are placed after the script name and a colon. Group names must be preceded by 'Group:'. If multiple tables/groups are specified the table/group names must be separated by a plus (+) character. When multiple scripts are run each definition, as described above, is separated by a semicolon (the string describing the associations must be bounded by quotes in this case; alternately, multiple *execute* commands can be issued from the command line). Bounding the value with quotes allows spaces and the semicolon to be used	*None*
fail	Selects whether or not to display event log fail messages	"true" to display event log fail messages in the main application window; "false" to hide event log fail messages. The value text is case insensitive	true
host	Sets the name of the PostgreSQL server's host	PostgreSQL server host name. The host name is case sensitive	*Previous session's PostgreSQL host (*localhost *for the first use)*
import	Imports the table(s) from the specified file(s)	Quoted (single or double) string containing one or more export sub-commands. See paragraph 4.9.3.8 for details	*None*

Command		Description	Value	Default Value
import sub-commands	appendExistingFields	See paragraph 4.9.3.7 for the description of the **Append existing data fields** check box	"true" or "false"	false
	fileName	See paragraph 4.9.3.7 for details. The file name is required by the `import` command	Name of the file, including the folder path, containing the tables to import. The file extension determines the format ("csv", "eds", "json", or "xtce")	*None*
	openEditor	See paragraph 4.9.3.7. Ignored if the GUI is hidden	"true" or "false"	false
	replaceExisting	See paragraph 4.9.3.7 for the description of the **Replace existing tables** check box	"true" or "false"	false
	useExistingFields	See paragraph 4.9.3.7 for the description of the **Use existing field if duplicate** check box	"true" or "false"	false
laf		Sets the application look & feel	"Look and feel" name (e.g., "Nimbus", "Windows", etc.). The names are case sensitive	*Previous session's L&F (*Metal *for the first use)*
logPath		Sets the path to the folder where the session event log is stored. This path only applies to the current session, and does not alter the one stored in the program preferences	Path to the folder in which to store the session event log	*None (the folder in which the application is started is used)*
mainSize		Sets the main application window's size	Main application window size in pixels. The parameter format must be in the form *width*x*height* where *width* and *height* are positive integer values. A width or height less than the minimum allowed (750 for width, 400 for height) is replaced by the minimum value	750x400
password		Sets the user's PostgreSQL password	Password for user name for PostgreSQL. The password is case sensitive	*None*
port		Sets the port of the PostgreSQL server's host	PostgreSQL server port. The server port must be blank or a positive integer	*Previous session's PostgreSQL port (*5432 *for the first use)*

Command	Description	Value	Default Value
project	Selects the project database to which to initially connect	Project database name. The project's database name is case sensitive	*Previous session's project name (none for the first use)*
scriptOutPath	Sets the path to the folder where the script output files should be stored. The scripts can access this path and use it to set the folder for the output files. This command may be used more than once in order to set the path for different `execute` commands. The path remains in effect for the `execute` commands until another `scriptOutPath` command is used to change it. This path only applies to the current session, and does not alter the one stored in the program preferences	Path to the folder in which to store the script output files	*None (the folder in which the application is started is used)*
server	Selects whether or not to display event log web server messages	"true" to display event log web server messages in the main application window; "false" to hide event log web server messages. The value text is case insensitive	true
shutdown	Disables display of the GUI and forces the application to exit following completion of the command line commands. This command is provided for use with running script associations, and importing or exporting tables via the command line options so that these operations can be performed without user interaction	*None*	*None*

Command	Description	Value	Default Value
ssl	Enables or disables a secure socket layer (SSL) connection to the PostgreSQL server. SSL must be set to "on" if the server expects an SSL connection, and "off" if it does not	"off" to disable SSL; "on" to enable SSL. The value text is case insensitive	*Previous session's SSL state (*off *for the first use)*
status	Selects whether or not to display event log status messages	"true" to display event log status messages in the main application window; "false" to hide event log status messages. The value text is case insensitive	true
success	Selects whether or not to display event log success messages	"true" to display event log success messages in the main application window; "false" to hide event log success messages. The value text is case insensitive	true
user	Sets the user name to use when connecting to the PostgreSQL server	User name for PostgreSQL. The user name is case sensitive	*Previous session's user name (none for the first use)*
version	Displays the CCDD version number and build date, then terminates the application. Any other commands are ignored		
webport	Set the port for the embedded web server. See paragraph 4.9.1.6.2 for more detail	Valid port number for the web server to listen to for queries	*Previous session's web server port (*7070 *for the first use)*
webserver	Enables the embedded web server. See paragraph 4.9.1.6.1 for more detail	"nogui" to start the application and enable the web server without displaying the user interface; "gui" to start the application, enable the web server, and display the user interface	nogui *or* gui

Table 1. Command line arguments

The following is an example of starting the application in Linux. In this example the CCDD application is installed in the current folder, the default libraries in the `CCDD_lib` folder are used, and the script library files are installed in the folder `/opt`. The project initially opened is "myProject" by user "userName":

```
java -classpath ./CCDD.jar:/opt/jruby-9.0.1.0/lib/jruby.jar:/opt/jython2.7
   .0/jython.jar:/opt/groovy-2.4.4/lib/groovy-2.4.4.jar:/opt/groovy-2.4.4/
```

```
lib/groovy-jsr223-2.4.4.jar:/opt/scala-2.12.4/lib/scala-compiler.jar
CCDD.CcddMain -project myProject -user userName
```

To make execution easier an alias can be created. Using the example above the Linux `alias` command is as follows:

```
alias CCDD='java -classpath ./CCDD.jar:/opt/jruby-9.0.1.0/lib/jruby.jar:/o
   pt/jython2.7.0/jython.jar:/opt/groovy-2.4.4/lib/groovy-2.4.4.jar:/opt/g
   roovy-2.4.4/lib/groovy-jsr223-2.4.4.jar:/opt/scala-2.12.4/lib/scala-
   compiler.jar CCDD.CcddMain'
```

For Microsoft Windows, the `doskey` command can be used to create an alias (the individual class paths must be separated by semi-colons instead of colons):

```
doskey CCDD=java -classpath "class_paths" CCDD.CcddMain $*
```

Having created an alias, the application can then be started by simply typing:

```
CCDD [args…]
```

An invalid command or command parameter results in program termination. An invalid parameter displays an error message at the command prompt. An invalid command or a valid command without an associated parameter produces the following output at the command prompt:

```
usage:
java -classpath <class_paths> CCDD.CcddMain [[<- or />]<command> <value> [...]]
Command line arguments:
  Command                 Value                              Description
  ----------------------  ---------------------------------  -------------------------------------
  backup                  backup file name                   Backup project on connecting
  command                 true or false                      Show command events
  events                  true or false                      Show events
  execute                 [association name] or              Execute script(s)
                          [" or ']script file name[:table1
                            or Group:group1[+...[+tableN or
                            Group:groupN]]][;...][" or ']
  export                  '<export sub-commands>'            Export tables, etc. in CSV, EDS, JSON,
                                                               or XTCE format

   export sub-commands:
   --------------------
    classification1       text (default: DOMAIN)             Classification level 1 (XTCE)
    classification2       text (default: SYSTEM)             Classification level 2 (XTCE)
    classification3       text (default: INTERFACE)          Classification level 3 (XTCE)
    endianess             big or little (default: big)       Endianess (EDS, XTCE)
    externalFileName      external export script file name   External export script file name
                                                               (EDS, XTCE)
    filePath              export file name                   Export file path + name (required for
                                                               EDS, XTCE, and for CSV, JSON if
                                                               exporting to a single file). This
                                                               path is in effect for the current
                                                               session only
    format                csv, eds, json, or xtce            Export file format
                            (default: csv)
    includeProjectFields  true or false (default: false)     Include project data fields
    includeReservedMsgIDs true or false (default: false)     Include reserved message ID list
                                                               (CSV, JSON)
    includeVariablePaths  true or false (default: false)     Include variable path list (CSV, JSON)
    isHeaderBigEndian     true or false (default: true)      Force telemetry & command header to
                                                               big endian (EDS, XTCE)
    overwriteFile         true or false (default: false)     Overwrite existing file(s). If the
```

```
                                                    GUI is hidden then any existing
                                                    files are always overwritten
  replaceMacros          true or false (default: false)     Replace macros with values
  singleFile             true or false (default: false)     Store in single file (CSV, JSON)
  tablePaths             <table1 or Group:group1[+...       Table paths (required)
                           [+tableN or Group:groupN]]
                           [;...]>
  validationStatus       text (default: Working)            Validation status (EDS, XTCE)
  version                text (default: 1.0)                Version (EDS, XTCE)
fail                     true or false                      Show fail events
host                     host name                          Set PostgreSQL server host
import                   '<import sub-commands>'            Import tables, etc. from a CSV, EDS,
                                                              JSON, or XTCE file

 import sub-commands:
 --------------------
  appendExistingFields   true or false (default: false)     Append existing data field(s) if
                                                              table exists. Only used if
                                                              replaceExisting is true
  fileName               import file name                   Import file name (required)
  openEditor             true or false (default: false)     Open an editor for each imported table
  replaceExisting        true or false (default: false)     Replace existing table(s)
  useExistingFields      true or false (default: false)     Use existing data field if imported
                                                              one matches. Only used if
                                                              replaceExisting and
                                                              appendExistingFields are true
laf                      look & feel                        Load look & feel
logPath                  file path                          Set event log file path. This
                                                              path is in effect for the
                                                              current session only
mainSize                 widthxheight                       Set main window size
password                 user password                      Set user password
port                     port number                        Set PostgreSQL server port
project                  project name                       Select CCDD project
scriptOutPath            file path                          Set the script output file path. This
                                                              command may be used more than once.
                                                              This path is in effect for the
                                                              current session only
server                   true or false                      Show web server events
shutdown                                                    Shutdown the application after
                                                              completing the command line
                                                              commands (e.g., script
                                                              execution(s), table imports, or
                                                              table exports. The GUI is not
                                                              displayed
ssl                      on or off                          Enable/disable SSL
status                   true or false                      Show status events
success                  true or false                      Show success events
user                     user name                          Set user name
version                                                     Display CCDD version
webport                  port number                        Set web server port
webserver                nogui or gui                       Enable web server
```

Once the application is executed the CCDD main window appears as shown in Figure 2. If password authentication is enforced (see paragraph **Error! Reference source not found.**) and a password is not supplied on the command line then the Select User dialog appears (see paragraph 4.9.1.1), allowing the

user and password to be entered. The graphical user interface (GUI) "look and feel" (L&F) can be selected by the user from a list of those installed on the operating system. If the L&F is changed then the application window and dialogs may differ in appearance (but not function) from those shown in the figures below. See paragraph 4.9.1.7.1 on how to alter the L&F.

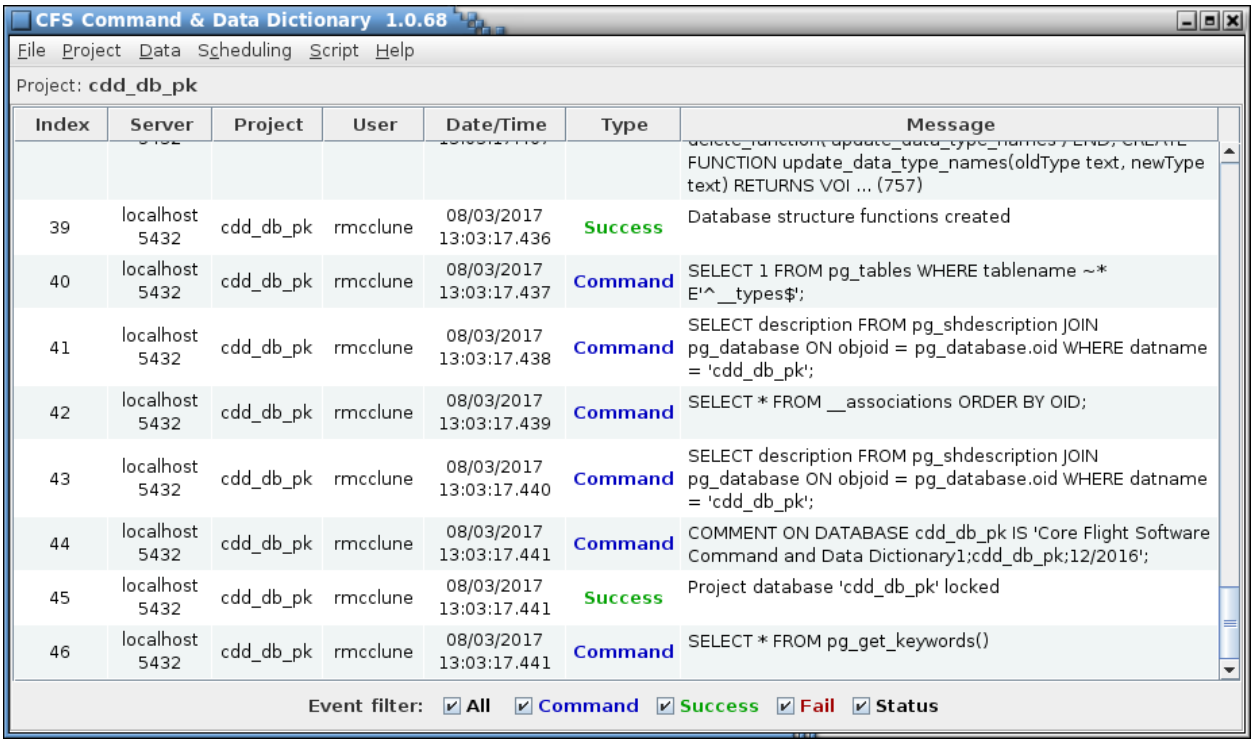


Figure 2. CCDD main window

The main window header contains the program name and version number. The main window is divided into a menu bar along the top and a session event log display area underneath. See paragraph 4.2.5 for further information on the event log. The window can be resized as desired. Each menu contains one or more menu items or sub-menus. A menu item that is grayed-out indicates that the affected item is not available at that time; for example, if no project database is open then the table commands are not available. A description of each of the menu items is provided in section 4.9.

4.2 Database

4.2.1 Setup

CCDD interacts with a PostgreSQL database. A description of installing and configuring the PostgreSQL software and server is beyond the scope of this document; see the PostgreSQL web site at www.postgresql.org for this information.

4.2.2 Authentication

The password authentication configuration for the PostgreSQL server affects the behavior of the CCDD application. Password authentication is controlled via the `pg_hba.conf` file. Super user status is required in order to make changes to this file. The location of this file can be determined by executing the "`SHOW hba_file;`" command in the PostgreSQL server command line utility **psql**. There are a number of authentication methods described in the documentation on the PostgreSQL web site; e.g. "trust", "password", "md5", etc. The methods can be applied to all users or to individual users based on the connection type (local or remote). When set to "trust" no password is required to log into the server and access a database. The methods "password" and "md5" are similar in that the user must

provide a password to log into the server. This is preferred in multi-user scenarios to control who may access the server and databases.

4.2.3 Roles

PostgreSQL allows only the owner of a database element (table, sequence, etc.) to make changes to that object. This would be problematic if multiple users require the capability to make updates. The restriction is overcome by means of *roles*. Every user login is a role in the server. Group roles can be created to which other roles (e.g., users) are assigned membership; any role belonging to the group inherits the privileges assigned to the group role. The PostgreSQL administrator must create a role for each user (the user's login identity), and one or more group roles that are used as the owner role when a project is created. The administrator must also assign membership in the group role to the appropriate users. Role creation and maintenance is performed outside the CCDD application, and must be completed prior to creating a project database. When a project database is created, one of the group roles is assigned as the owner (see paragraph 4.9.2.3; note that for a single-user project the user's role can be selected as the owner). Since all elements of the database are owned by the selected group, all members of the group have write privileges to these elements. Other roles (users) not in the group are prevented from changing the project elements. To control the ability of a specific user within the group to alter the database, CCDD implements individual user access levels – see paragraph 4.2.5.

4.2.4 Connection recovery

If a database transaction fails due to a lost connection an attempt is made to reestablish the connection. If successful then the transaction is automatically repeated. If the reconnection attempt is unsuccessful then the error condition is propagated to the calling method. The PostgreSQL server timeout value, used when attempting to reestablish the server connection, is a program preference value (see Appendix E.2).

4.2.5 Project access control

It's desirable sometimes to allow users access to the data within a project, but not allow these users to alter the data. CCDD implements user access levels to accomplish this. *This is intended only as a low level security measure to prevent inadvertent alteration of project data and does not prevent access to a project's database and/or alteration of the data via other applications*. The three access levels are administrative, read/write, and read only. When a project is created the user creating the project is automatically granted administrative privileges. Users with administrative access can assign access levels to the other users associated with the project. A user with read only access can still alter the contents of any of the editors in CCDD, but is unable to store the changes. Users granted read/write access have full access to the project, but are not allowed to alter access levels for any user. The enable state or operational capability for a command menu item is based on the user's access level.

Following project creation only the project's creator has a defined access level. Users that are not explicitly assigned an access level have read only privileges. Therefore, all other users with a role in the project can't alter the project's data until the project's administrator assigns them read/write or administrative privileges via the user access level manager - see paragraph 4.9.2.11.

4.3 Event Log

The application automatically records all interactions with the PostgreSQL and web servers. The information includes the exact commands issued to the server and the server responses (success, or failure with supporting information). All events are logged to the session's log file, even if the GUI is disabled.

When the GUI is enabled the main application windows displays the current session's event log. Previous sessions' event logs can be reviewed using the **Read log** command; see paragraph 4.9.1.3. The log automatically scrolls to the latest entry when an event is logged. Each log entry contains the following information arranged in a tabular format:

Index This is a sequential number assigned to each log entry, beginning with 1 and incrementing by one as transactions occur with the database.

Server Name or address of the computer hosting the PostgreSQL server and the port number.

Project Name of the project database to which the transaction applies.

User Name of the user that initiated the transaction.

Data/Time Date (month/day/year) and time stamp (hours:minutes:seconds) when the transaction occurred.

Type One of five log entry types:

- **Command** Indicates a PostgreSQL command issued to the database.
- **Success** Indicates the database transaction completed successfully.
- **Fail** Indicates the database transaction or web server command failed.
- **Status** Indicates the log entry provides application status information.
- **Server** Indicates a web server command.

Message The text in this column is dependent on the message **Type**. For a **Command** type the text is the PostgreSQL command issued to the database. If a data base transaction succeeds then a **Success** type message indicates what was accomplished by the database command. A **Fail** type message provides details on the cause of the transaction or web server failure. Failed transactions are rolled back so that no change is made to the database. The **Status** type message shows the results for an application operation (for example, the database table consistency check). A **Server** command displays web server command information. The message length displayed is limited by default to 250 characters in order to prevent bogging down the application. Truncated messages are denoted by a trailing ellipsis (…) followed by the number of truncated characters in parentheses. The full text of the message can be viewed by double clicking the right mouse button while the mouse pointer is over a log entry row – a log entry viewer is opened showing the full message text for that row.

Beneath the logged entries are entry filter check boxes that can be used to determine which messages are displayed, based on the message type(s). If a message type's check box is unchecked then messages of that type are hidden. Checking the box restores the messages. Messages for hidden types are still logged even if not currently displayed. The **Server** check box only appears once the web server is activated. The **All** check box affects the other check boxes – unchecking it clears the other check boxes, and checking it selects the others. If none of the check boxes are selected no log entries are displayed. Note that for the single log entry viewer the filter check boxes are not displayed.

A large number of **Command** log entries are generated during normal program operation. The display of these events can slow program execution under some circumstances (for example, when exporting a large number of tables). Deselecting the **Command** check box hides the command entries, making the GUI more responsive. The check box can be reselected, causing the command entries to be displayed, if the entries need to be reviewed. A command line option can be used to deselect the check box at program startup.

4.4 Mouse and Keyboard Navigation

The application's menus, dialogs, and GUI components can be manipulated using the mouse pointer, mouse buttons, and mouse wheel, as well as with the keyboard. Keyboard mnemonics are provided for the menu items and dialog buttons. These are accessed by pressing the Alt key in addition to another key; i.e., Alt+*key*, where *key* is the underlined character in the menu or button text (the key case is ignored). For example, pressing Alt+F or Alt+f in the main application window opens the **File** menu. The Tab and arrow keys can be used to navigate between the components in a dialog or window, and the pressing the Enter or space key actuates a control (e.g., a button or check box). See Appendix E.1 for a list of special keys and key sequences.

When a dialog containing a table is initially displayed it has no table row selected. A row can be selected by positioning the mouse pointer over a cell in the row and pressing the left mouse button, or by using the keyboard. To select an initial row with the keyboard press the Tab key, then the Enter or Space key when the table has the keyboard focus (which it does initially); this selects the table's topmost visible row and sets the focus to that row's leftmost column. The up and down arrow keys can then be used to change the selected row and the left and right arrows can change the selected column. The selected cell is highlighted. Multiple cell selection behavior is dependent on the particular table, but in general behaves as follows. Multiple, contiguous cells can be selected using a combination of the mouse/keyboard and the Shift key. Highlight the starting cell, then either (a) continue to press the left mouse button and drag the pointer, (b) hold the Shift key and left-click the mouse on another row (the two rows, plus any in between, are highlighted), or (c) hold the Shift key and press the arrow key to highlight as many cells as desired. Individual cells can be selected/deselected by pressing the Ctrl key and selecting the cell with the mouse. The entire table may be selected by pressing Ctrl-A. For row operations (e.g., Move up or Delete row) the row(s) indicated by the highlighted cell(s) are affected. Similarly, for column operations (e.g., Move left) the column(s) indicated by the highlighted cell(s) are affected. Once one or more cells are selected the highlighted data can be copied by pressing Ctrl-C. To paste the data into another application (e.g., spreadsheet or text document) or another table use the Ctrl-V or Ctrl-I key sequence.

Navigation within a table can be accomplished via mouse or keyboard. Note that some of these keys perform different functions if a cell is actively being edited. The Insert key inserts a row at the current selection point and the Delete key erases the contents of the currently selected cell(s) (see above paragraph concerning cell selection). Pressing the Ctrl-Delete deletes the currently selected rows. The Home and End keys change the cell selection to the first or last column, respectively, of the currently selected row. The Page Up and Page Down keys scroll the table up or down one page, respectively, (unless the entire table is already visible) changing the cell selection to the currently selected column, with the row one page up or down from its previous position.

Table data entry is initiated by double clicking the left mouse button while the mouse pointer is over the cell to be edited. The Enter or Space keys may also be used to initiate editing on the currently selected cell (the Space key initiates editing as well as inserts a space into the cell at the end of any existing text). Pressing the Enter key while editing a cell stores the text in the cell and initiates editing in the next cell, moving left to right until the last column is reached, at which point editing moves to the first column in the next row below unless the end of the table is reached. Protected cells, denoted by a gray background color, are skipped. A cell containing a check box does not allow moving to the next cell via the Enter key; instead, the check box state is toggles with each press of the Enter key.

While cell editing is active the Insert key inserts a space to the right of the text cursor, and the Delete key deletes the character to the right of the text cursor. The Home and End keys move the text cursor to the beginning and end of the cell, respectively. If the table cell's input type (see 4.7 for information on input types) supports multiple lines then pressing Alt-Enter inserts a line break.

Pressing the Escape key while editing terminates editing of the cell and removes any changes made to the cell.

For most tables in the application, row sort order, column width, and column position are user-adjustable. The table rows can be sorted by column by positioning the mouse pointer over the column's header and pressing the left mouse button. The rows are sorted in ascending order, depending on the selected column's contents, and an icon appears beside the column name indicating the sort direction. Selecting the column again sorts in descending order (with a corresponding change in the sort direction icon), and a third selection restores the rows to their original order and removes the sort direction icon (further column header selection repeats this sequence). Only one column can be sorted at a time – selection of another column removes the sort from the first sorted column before applying the sort to the newly selected one. The column width may be resized by positioning the mouse pointer over the right border of the column header (the mouse pointer changes to indicate resizing is possible), pressing and holding a mouse button, then moving the mouse left or right; release the mouse button to exit resizing. Automatic resizing, based on the widest of the contents of the cells and header, is accomplished by double clicking a mouse button when the resize cursor appears. The column order may be changed by positioning the mouse pointer over a column header, pressing and holding the left mouse button, then dragging the column to the new location.

Certain dialogs contain a tabbed pane and multiple tabs (example, the table and link editors). The tab order can be rearranged by positioning the mouse pointer over the tab, pressing and holding the left mouse button, and moving the mouse pointer. The pointer icon changes and a transparent copy of the tab appears that follows the pointer, but is constrained by the tabbed pane's header area. As the copy is moved the insertion point is indicated by a thicker line appears beside the tab where the dragged tab will be moved if the mouse button is released. For the table editor dialog, if the pointer is moved outside the bounds of the dialog and the mouse button released then the table editor represented by the tab is removed from the editor dialog. If the pointer is over another editor dialog then the tab is added to that dialog at the position indicated by the insertion indicator. However, if the button is released while the pointer isn't over a table editor dialog then a new editor dialog is created and the tab is placed within it.

Details specific to navigation in certain windows and dialogs are provided in the components' descriptions in later sections.

4.5 Data Tables

The CCDD data is stored in the project's database in the form of tables. The tables consist of a two-dimensional array of columns and rows. The columns define the content of the cell in each row, much like the data in a spreadsheet. For example, a table may have a column titled "Description" which indicates that the cells in that column contain descriptive text concerning the parameter defined in each specific row. There is no constraint on the number of tables in the project's database, nor is there a constraint on the table's number of columns and rows.

4.5.1 Table types

Every data table is built from a table type definition which defines the table's columns. Think of a table type as the blueprint from which other tables are created. Two types of tables are available by default upon creation of a project: *Structure* and *Command*. Structure tables represent C-program data structures containing information on variables. Command tables are designed to contain information pertinent to CFS commands. Other table types may be created by the user to contain data that doesn't fit into the predefined types (see paragraph 4.9.3.10 for information on the table type editor). All tables of a given type share the same column definitions. Data in tables of any type are accessible via the scripts (see paragraph 4.10 for information regarding script access).

Every table that is created from a table type is considered a *prototype*. A prototype determines the columns and default data for all *instances* of that table. Each prototype table itself constitutes an instance of that table, and in many cases the prototype is the only instance. However, in the case of structure tables, multiple instances can exist – one for every reference to the structure from within another structure. Each of these derive their columns and initial data values from their prototype table.

A prototype table that is not referenced from within another table is considered a *root* table. A table that references another table is that table's *parent*, and the referenced table is a *child* of the parent. The root table is also a parent if it contains a reference to another table. It's common for structure tables to have a parent-child relationship. It's possible for tables of other types to have such a relationship as well, though less likely.

4.5.1.1 Structure tables

Structure table rows represent C-program variables and related information. The variables can either be of a primitive data type (e.g., integer, char, double) or can be a reference to another structure. These child structures can in turn reference other structures, and so on, to any depth required by the user. The only constraint is that no circular references are allowed, wherein a structure references itself somewhere in its hierarchy. Ultimately only references to primitive data types exist as the end point of any path from the root structure, through its child structures, to a variable.

Certain columns are inherent to structures and must be present for the table to be recognized as a structure. The default names for these columns are "Variable Name", "Data Type", "Array Size", "Bit Length", and "Rate". The column names can be changed if desired; it's the column's input type that identifies the column (see paragraph 4.7 for more information on input types). Therefore, for a table to be treated as a structure it must include at a minimum the columns with the input types "Variable name", "Primitive & Structure", "Array index", "Bit length", and "Rate". Other columns, "Description", "Units", and "Enumeration", are automatically included for structure tables; these additional columns can be altered, or even deleted. Columns containing other variable information can be added at the user's discretion. Table 2 shows the default structure column names and the corresponding input types.

Default Column Name	Input Type
Variable Name	Variable name[1]
Description	Description
Units	Units
Data Type	Primitive & Structure[1]
Array Size	Array index[1]
Bit Length	Bit length[1]
Rate	Rate[1]
Enumeration	Enumeration

1 *A table must contain this input types to be identified as a structure*

Table 2. Structure column names and input data types

Only one variable name, data type, array size, and bit length column is allowed per table type definition. The table can have multiple rate and enumeration columns.

The array size and bit length cell values are mutually exclusive for a variable; only one can be assigned (or neither). If array size is specified then the bit length cell for that variable is grayed out and cannot be selected. Conversely, if a bit length is entered then the array size cell is grayed out.

The bit length and enumeration cells are valid only for data types that have a base type of "signed integer" or "unsigned integer" (see paragraph 4.5.4 for information on data types). If a non-integer data type is selected in the data type column then the bit length and enumeration cells for that row are grayed out and cannot be edited. Conversely, if the bit length or enumeration cell is not empty then the data type for that row only displays integer data types.

If an array size is specified a row is inserted automatically into the table for each array member. Arrays may have one or more dimensions, each with a minimum size of 2. For multi-dimensional arrays the size of each dimension is specified in the array size column, separated from one another by commas. For example, a three dimensional array, **n**, with dimension sizes of 2, 3, and 4 would have the array size specified as "2, 3, 4" (the array text is automatically formatted with spaces). The first array member would be **n[0][0][0]**, the second **n[0][0][1]**, and so on until the last member, **n[1][2][3]**, is reached; this array would have a total of 24 members (= 2 x 3 x 4). When a structure table is open any arrays are initially collapsed; in other words only the *array definition* row is shown. The display of the *array member* rows can be toggled in one of two ways, via the **Expand arrays** command (see paragraph 4.9.3.2.3.5) or by positioning the mouse pointer over any cell in the array size column and double-clicking the right mouse button. When expanded, the array members for all arrays are displayed beneath their respective array definition row. The variable name column shows the variable name with the array index (or indices) appended, and the overall array dimension size(s) is displayed in the array size column. See Figure 3 for an example of an expanded array definition.

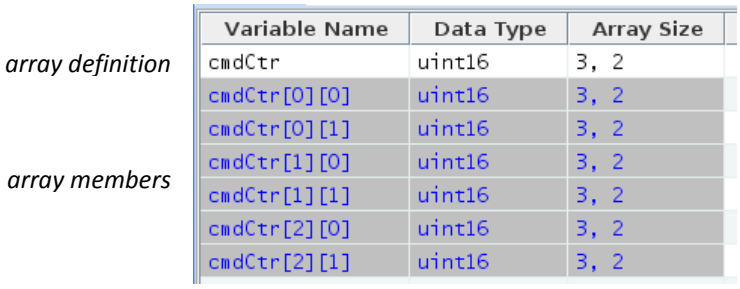


Figure 3. Example array display

Note that the variable name, data type, and array size are grayed out and cannot be altered in the array member rows; however, individual values may be assigned to a member for the other columns in the table. The exception is the bit length column – an array definition or member cannot be assigned a bit length. To change the array member names or data type make the change to the array definition row; the member rows are changed as well. Changing the array definition row's array size value increases or decreases the member rows as needed, and clearing the array size cell removes all of the member rows for that variable.

The string data type is a special instance of the character base data type. If no array size is specified then the string variable is simply a single character. If an array size is supplied then the first (rightmost) array dimension determines the string length in characters. A string behaves as other array variables except that column values (e.g., description) may not be assigned to array members of the string other than the first one (i.e., one ending with an array index of zero). Arrays of string variables are allowed, as with other data types.

If a data type cell references a structure then the specific instance of the structure table it represents can be opened by double-clicking the right mouse button while the mouse pointer is positioned over the data type cell. The table is opened in its own tab in the same editor window (see paragraph 4.9.3.2 for

more information on the table editor). If this is attempted on a structure reference in a prototype table, and the prototype table is itself referenced in another structure table, then the dialog in Figure 4 is displayed, indicating that the prototype of the selected structure, and not a specific instance, was opened ('*a_structure_table*' and '*a_child_table*' in the figure are replaced by the prototype and child table names respectively). Once a structure is referenced by another one it is no longer a root structure table, and therefore can't have its own child tables, only those that are part of the hierarchy of the root structure to which the structure's prototype belongs.

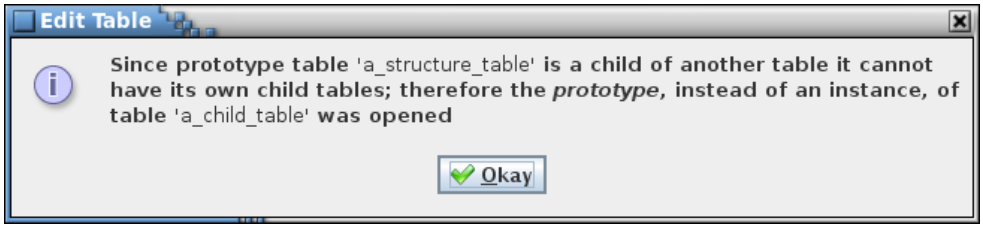


Figure 4. Example dialog

As mentioned above, a child table inherits the data values of its prototype table. If a child table's data value is edited it overrides the inherited value, producing a "custom" value for the child. Even if the child's prototype values are subsequently changed the child retains its "custom" values. Special action must be taken to remove the child's custom value and have it use the prototype value – see paragraph 4.9.3.2.2.8.2.

The user may create other table types that also represent a structure table. If a table contains the default structure input data types then the table is treated as a structure table.

4.5.1.2 Command tables

Command tables contain CFS command information. Certain columns are inherent to command tables and must be present for the table to be recognized as a command table. The default names for these columns are "Command Name" and "Command Code". The column names can be changed if desired; it's the columns' input type that identifies the column (see paragraph 4.7 for more information on input types). Therefore, a command table must include columns with the "Command name" and "Command code" input types. Other columns ("Description", "Arg 1 Name", "Arg 1 Description", "Arg 1 Units", "Arg 1 Data Type", "Arg 1 Array Size", "Arg 1 Enumeration", "Arg 1 Minimum", and "Arg 1 Maximum") are automatically included for command tables; these additional columns can be altered, or even deleted. Only a single command name and command code column is allowed per table type definition. Columns containing other command information, such as those to describe more command arguments (name, data type, enumeration, minimum, maximum, etc.) can be added at the user's discretion. Table 3 shows the default command column names and the corresponding input types.

Default Column Name	Input Type
Command Name	Command name[1]
Command Code	Command code[1]
Description	Description
Arg 1 Name	Argument name
Arg 1 Description	Description
Arg 1 Units	Units
Arg 1 Data Type	Primitive
Arg 1 Array Size	Array index
Arg 1 Enumeration	Enumeration
Arg 1 Minimum	Minimum
Arg 1 Maximum	Maximum

1 *A table must contain this input types to be identified as a command*

Table 3. Command column names and input data types

The order in which the columns are defined in the command table type definition determines which columns are associated with a particular command argument. See paragraph 4.9.3.10 for details on defining columns for a table type definition. Any column defined prior to the first column with the input type "Argument name" is assumed to apply to the command in general. All column definitions between two columns with the input type "Argument name" (or the last argument name column and the end of the table type definition) are assumed to be associated. For the default command table type definition the Description column is considered generic, and all of the columns following "Arg 1 Name" are associated with the first (and only) command argument. If a subsequent command argument is added it would begin with a column of input type "Argument name", placed after all of the first argument's associated columns, and followed by any column definitions associated with the new argument. It is suggested that any extra command argument sets are named in a manner similar to the default argument 1 names (e.g., Arg 2 Name, Arg 2 Data Type, etc.) so that the association is apparent to the user.

The command argument enumeration cells are valid only for data types that have a base type of "signed integer" or "unsigned integer" (see paragraph 4.5.4 for information on data types). If a non-integer data type is selected in the argument's data type column then the argument's associated enumeration cell in the same row is grayed out and cannot be edited. Conversely, if the argument's enumeration cell is not empty then the argument's associated data type for that row only displays integer data types.

4.5.2 Table groups

Data tables can be assigned to user-defined groups (see paragraph 4.9.3.9 for details on assigning tables to a group). These groups are a method of relating tables to each other. For example, all of the tables for a specific CFS application or subsystem can be assigned to a group. The groups are used in filtering the table tree (see paragraph 4.5.3 concerning tree filtering). A table can be assigned to more than one group, or to none. The application scheduler uses the groups designated as CFS applications when producing the scheduler table (see paragraph 4.9.3.9 for details on designating a group as an application, and paragraph 4.9.4.3). Groups can also be assigned data fields. If a group is specified as

representing a CFS application a number of data fields are automatically assigned (these can be edited, removed, or additional fields added as desired). See paragraph 4.6 for more details on data fields.

4.5.3 Table tree

The table tree displays the data tables using a tree representation. Depending on the operation (e.g., Edit, Rename, etc.) there are one or two top level branches in the tree. The first, labeled *Prototypes*, is an alphabetical arrangement of the prototype tables. Since it displays prototypes only it is a single level in depth (not including any filtering; see below). The second top level branch that may be displayed, *Parents & Children*, shows the root tables and, if applicable, their children as sub-branches, and the children of those tables as further sub-branches, etc. See Figure 5 and Figure 6.

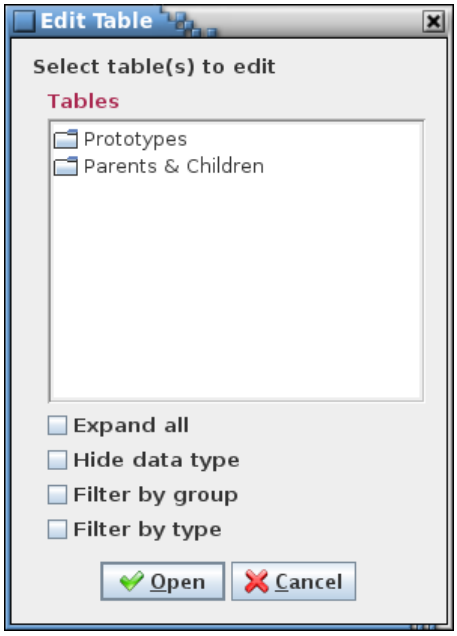


Figure 5. Table tree

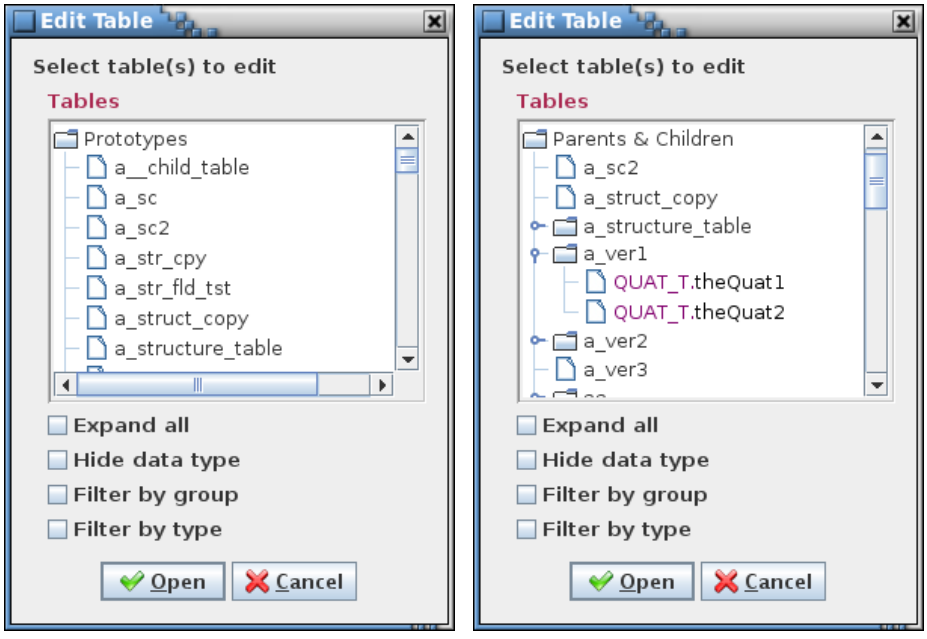


Figure 6. Table tree expansion

Selecting the symbol beside a branch in the tree causes that branch to expand (if collapsed) or to collapse (if expanded). Selection can be made with the mouse pointer, or by using the tab key and up/down arrows to highlight the branch's name, then pressing the right arrow to expand or left arrow to collapse the branch. Positioning the mouse pointer over a branch name and double left-clicking toggles between expanded and collapsed view for that branch. Selecting one or more branches and pressing Ctrl-E causes the selected branches and all of their child branches to expand (if collapsed) or collapse (if expanded). The first branch selected determines if any other selected branches are expanded or collapsed. Hovering the mouse over an item in the tree displays a pop-up tool tip showing the description of the item (if it has one; for a table this is the text from the table's description field; see Figure 35).

Below the tree are one or more check boxes. The **Expand all** check box is available for every tree; selecting this check box causes all of the tree branches to be displayed. Clearing the check box collapses all of the branches down to the initial level.

The names displayed in the tree for structure and primitive variables are in the format *data type.variable name* (see paragraph 4.5.3.1). For structure variables the data type is the name of the structure defining the variable; for primitive variables (e.g., integers, floats) the data type is the name of the primitive. Color is used to highlight the data type. The **Hide data type** check box appears with table trees that display structure and primitive variables. When unchecked, the tree displays the variables in the full format as described. When check, the data type portion of the name is not displayed.

The remaining check boxes are used to filter the tree contents. There are two filter methods, **by groups** and **by types**. Depending on the operation one, both, or neither of these check boxes may be available.

If the **Filter by group** check box is selected then sub-branches are inserted at the level below the *Prototypes* and *Parents & Children* branches. These sub-branches are the groups defined by the user (see paragraphs 4.5.2 and 4.9.3.9). Tables belonging to the group are displayed as sub-branches of the group branch. A special group, labeled "*All tables*", appears in the tree below the user-defined group sub-branches. The "*All tables*" group is an automatically defined group that displays all tables, including those that are not a member of a group, so that every table is still available for selection in the table tree while the group filter is applied. Deselecting the check box removes the group branches.

If the **Filter by type** check box is selected then sub-branches are inserted at the level below the *Prototypes* and *Parents & Children* branches. These sub-branches are the table types: structure, command, and any others defined by the user (see paragraphs 4.5.1 and 4.9.3.10). Tables of a given type are displayed as sub-branches of the table type branch. In other words, all of the Structure type tables appear under a *Structure* branch, all Command type tables under a *Command* branch, and so on for each defined table type. Deselecting the check box removes the type branches.

Both the group and type filters may be applied simultaneously. The branches are first divided by group. Each group is then sub-divided by table type.

4.5.3.1 Variable tree

Another form of the table tree is the variable tree. The variable tree displays only the project's structure tables. These are displayed in the same manner as in the table tree, except that the variables belonging to the structure tables are also shown as branches of their parent structure. Variable trees are used where selection of variables is required; e.g., in the links manager (paragraph 4.9.4.1) and the telemetry scheduler (paragraph 4.9.4.2). Like the table tree, variable trees allow filtering by group.

Variable names are displayed in the tree in the format:

<*data type*>.<*variable name*[*[array size][…]*]>[*:bit length*]

Examples: float.bq[1], uint16.faultBits:12

The node icons used in the variable tree indicate if the variable is a bit-wise variable or not (i.e., has a bit length assigned), if the variable is bit-packed with one or more variables, and if the variable belongs to a link (see Table 4). Paragraph 4.5.5 provides details on bit-packed variables.

Icon	Variable type
	Non-bit-wise variable
	Linked non-bit-wise variable
	Bit-wise variable
	Linked bit-wise variable
	Packed bit-wise variable
	Linked and packed bit-wise variable

Table 4. Variable tree icons

4.5.4 Data types

The structure and command tables, and possibly and user-defined table types, contain data type columns. This column is used to set the data type for the referenced parameter (e.g., structure variable or command argument). The data type is either a primitive type, a reference to a structure, or a pointer.

Each primitive data type is derived from one of five base data types: signed integer, unsigned integer, floating point, character, and pointer. The base type, along with the size in bytes, determines the characteristics and usage of the data type. For example, a bit length can be assigned to a variable only if its data type has an integer base type (signed or unsigned), and the bit length is less than or equal to the data type's size (in bits).

A project database is automatically provided with a number of primitive data types. These can be altered or deleted. The default primitive types are shown in Table 5.

Data Type Name	C-Language Data Type	Number of Bytes	Base Type
int8_t	signed char	1	signed integer
int16_t	signed short int	2	signed integer
int32_t	signed int	4	signed integer
int64_t	signed long int	8	signed integer
uint8_t	unsigned char	1	unsigned integer
uint16_t	unsigned short int	2	unsigned integer
uint32_t	unsigned int	4	unsigned integer
uint64_t	unsigned long int	8	unsigned integer
float	float	4	floating point
double	double	8	floating point
char	char	1	character
string	char (array)	>1	character

address	void *	4	pointer

Table 5. Default primitive data types

A data type with a base type of 'character' is considered a string if the byte size is set to greater than 1. The byte size value in this case is otherwise unused by the application. The application treats a string as an array of characters. An array of data type 'string' is treated specially by the application. See paragraph 4.5.1.1 for details.

To the application a pointer represents an address and the actual data type and C type names are irrelevant. However, the application does allow creation of pointers with distinctive names. This is useful, such as in a generated header file to create `typedef` statements for subsequent use in assigning data types to variables (versus using the `void *` data type and type casting each variable appropriately). The application allows creation of any number of pointer data types.

In a data table, when a data type column cell is selected it displays a drop down menu showing the data types. The data types available depend on the usage. In general, in a structure table the data types include primitive types and the names of structures that are not referenced in the hierarchy of the structure being edited (this prevents creating a circular reference). If the structure variable has a bit length or enumeration value then the data types available are limited to primitive types with an integer base type (signed or unsigned). For a command argument only primitive types are displayed, and if the argument has an enumeration value then the data types are limited to primitive types that have an integer base type (signed or unsigned).

The data type manager (paragraph 4.9.3.11) is used to create, modify, and delete the primitive data types.

4.5.5 Bit fields

Variables with an integer (signed or unsigned) data type may be assigned as bit fields. A bit field is identified by having a value entered in the structure data table's **Bit Length** column. Variables with bit lengths specified that are co-located in the table and have the same data type are assumed to be packed together; i.e., these variables occupy the same byte or bytes. The number of variables and bits that are packed is based on the data type's byte size and the bit length of each variable. The bits representing a variable must be contained within a single data type's set of bits. For example, a *uint16* is two bytes, or 16 bits, so bit field variables totaling 16 or fewer bits are packed. If three variables of type *uint16* are co-located, with bit lengths of 2, 12, and 5, then the first two variables are packed together (2 + 12 < 16), and the third variable occupies its own pair of bytes since its 5 bits won't fit within the first packed pair's 2 unused bits.

Bit-packed variables must have the same telemetry downlink rate. Since the variables are packed together they are downlinked together, even if only a subset of the variables is desired. The table editor accounts for bit-packing by enforcing a common rate among variables that are packed together. In other words, it changes the rates, if needed, of packed variables so that they match. The check for, and update to, a common rate takes place each time an edit is made to the table. In order to prevent two variables from being packed together a padding variable must be added between them with the appropriate bit length to ensure the two variables no longer fit within the bit size of the variables' data type.

When transferring variables, such as between trees in the link manager or between the variable tree and messages in the telemetry manager, those that are packed together are automatically moved as a unit, even if not explicitly selected.

4.5.6 Enumerations

Enumerations allow associating a text label with an integer value, and optionally other attributes. Enumerations are useful, for example, in displays, since descriptive label text can be substituted for an ambiguous numeric value.

The format for an enumeration is as follows:

<1st enum value> <value/label separator> <1st enum label> [*<value/label separator> <other 1st enum attributes>*]
[<pair separator> *<2nd enum value> <value/label separator> < 2nd enum label>* [*<value/label separator> <other 2nd enum attributes>*]][[*<pair separator>*…] [*<pair separator> <nth enum value> <value/label separator> < nth enum label>* [*<value/label separator> <other nth enum attributes>*]]]

The enumeration value/label and pair separator characters are at the discretion of the user. The application detects these characters automatically.

The label can contain multiple attributes (e.g., color, limit, etc.) – use the value/label separator character to delineate each attribute. Below is an example of an enumeration with three possible values (0, 1, and 2):

0 | Off | red, 1 | On | green, 2 | Standby | yellow

In this example the enumerated values 0, 1, and 2 correspond to the labels "Off", "On", and "Standby" and the colors "red", "green", and "yellow" respectively. The value/label separator is the "|" character, and the pair separator is the comma (","). Any spaces and/or tabs bounding the separator characters are ignored.

The structure and command tables contain an enumeration column by default. The enumeration's integer value is the value of the parameter described in the same row of the table (the variable name for a structure table and the argument name for a command table). The enumeration parameter's data type must be an integer type (signed or unsigned). The structure and command table editors enforce this constraint by not allowing text to be entered into an enumeration cell for which the associated data type is not an integer, and by only displaying integer types in the data type cell if the associated enumeration cell is not blank. Data type and enumeration column associations are determined by their respective input type designation (see paragraph 4.7) and are paired based on their order in the table's type definition. Command tables may have one or more command arguments, each with a data type and enumeration (see paragraph 4.5.1.2); these are automatically paired when both are present in a given argument. The EDS and XTCE XML conversions (see paragraphs 4.9.3.8.2 and 4.9.3.8.3) also check that the data type is valid for an enumeration, generating an error message if an enumeration is associated with a non-integer data type.

4.5.7 Macros

A macro is a text string used to represent a number or text. Once defined, a macro can be used to replace part or all of the contents of a data table cell. This allows a commonly used string of text to be defined once, then used in as many tables and table cells as desired. If the text subsequently needs to be altered then only the macro's definition need be changed, instead of having to locate and change each table cell where the text is found. An example for such usage would be an enumeration used in multiple cells and/or tables.

Macros are created and their values set or altered using the macro editor, described in paragraph 4.9.3.12. A macro name, when entered into a cell, must be bounded on either side by a pair of '#' characters (##, with no intervening spaces) in order for the macro to be recognized. Text that's entered into a cell bounded by the macro delimiters is only recognized as a macro if the macro is defined. If the macro isn't recognized then the characters are treated as any other text string. If the macro is

subsequently defined the cell automatically recognizes the text string as a macro. Macros can also be entered using the **Insert macro** command in the table editor; see paragraph 4.9.3.2.2.6.

A special type of macro is the *sizeof(data type)* call. The *sizeof()* call returns the size in bytes of the specified primitive or structure data type. *sizeof()* can be used in the same locations as a macro. The macro identifier characters (##) are not used with the *sizeof()* call.

Multiple macros can be inserted into a cell. However, a macro can't be inserted within another macro (the macro into which the second macro is inserted is no longer recognized as a macro in this case).

Text containing a macro and/or sizeof() call is evaluated as a mathematical expression if the resulting expansion of the macro(s) and *sizeof()* call(s) in the text is in a valid expression format. Mathematical expressions may contain numbers (including floating point values), the operators '+' (addition or sign, based on context), - (subtraction or sign, based on context), '*' (multiplication), and '/' (division), parentheses ('(' and ')') for nesting expressions, and spaces (which are ignored).

As an example assume two macros are defined, MACRO_A with a value of "2" and MACRO_B with a value of "3". The text "(##MACRO_A## + 4) / ##MACRO_B", when the macros are expanded, is "(2 + 4) / 3". Since this is a valid mathematical expression the resulting text is "2" since (2 + 4) / 3 = 2. Multiple expressions can exist in the same text if each is separated from the other by a comma (,). For example, the text "#MACRO_A, sizeof(float) * 3" when expanded is "2, 4 * 3" (using the value for MACRO_A stated previously and assuming a float is 4 bytes). Each portion of the text, when separated at the comma, is a valid mathematical expression, so the resulting text is "2, 12". This format is useful when setting the array size for a multi-dimensional array variable.

If the mouse pointer is hovered over a cell containing a macro a tool tip pop up appears displaying the contents of the cell with each macro name replaced by its value. All of the macros in a data table can be temporarily replaced by their corresponding values using the **Show macros** command in the table editor; see paragraph 4.9.3.2.2.7.

When a table's data is retrieved for use in a script or via the web server the option exists to retain the macro names in place of the macro values. See paragraphs 4.10 and 4.9.1.6 for details. An example of use for this is when creating C header files, where a `#define` statement is used to set a constant that determines array variable size(s). The macro name can be used to set the `#define` constant's name and value. In the array definition(s) the macro name is retained instead of the using the value so that the `#define` constant determines the array size (note that the macro delimiter characters must be removed in this example).

4.6 Data Fields

Data fields are input fields created by the user for entering information associated with the component to which the field belongs. The fields are assigned names and an input type that constrains the values that can be entered into the field. Data fields can be associated with a project's data tables, groups, and to the project as a whole.

A data field can be used to enter a piece of information for a data table that doesn't fit with a table's row and column format. An example is a message identification (ID) number for a root structure table – the message ID applies to the entire table, not a specific row within it. A column could be added for information such as the message ID, but having the same value repeated for each row is both wasteful in storage as well as prone to errors (if every value doesn't match). Any number of data fields (including none) can be associated with each table. Default data fields may be assigned to a table type, so that every table created of that type automatically has the default fields in place. Fields can also be assigned to individual tables – there is no requirement for the tables to have the same fields.

Similarly, table groups can be assigned data fields. For example, a group can be created that consists of all of the tables for a specific CFS application, so that the group represents the application. Applications have data associated with them that isn't appropriate for storage in a table, such as the application schedule rate or execution time. In this case a data field can be assigned to the group to hold the information. Groups designated as CFS applications are automatically assigned certain data fields; see paragraph 4.9.3.9.

Some data associated with a project is applicable to the entire project. Examples include the project's name and description, or the number of central processing units (CPUs) supported. For these cases the project-level data fields are appropriate. Paragraph 4.9.3.15 provides information on adding data fields at the project level.

4.6.1 Data field editor

This section provides details on use of the data field editor (Figure 7). See paragraph 4.9.3.10.4.1 for information regarding adding default data fields to table types, paragraph 4.9.3.2.5.1 for information on adding fields to a particular table, and paragraph 4.9.3.15 for information on adding project-level fields. In each case the editor operation is the same.

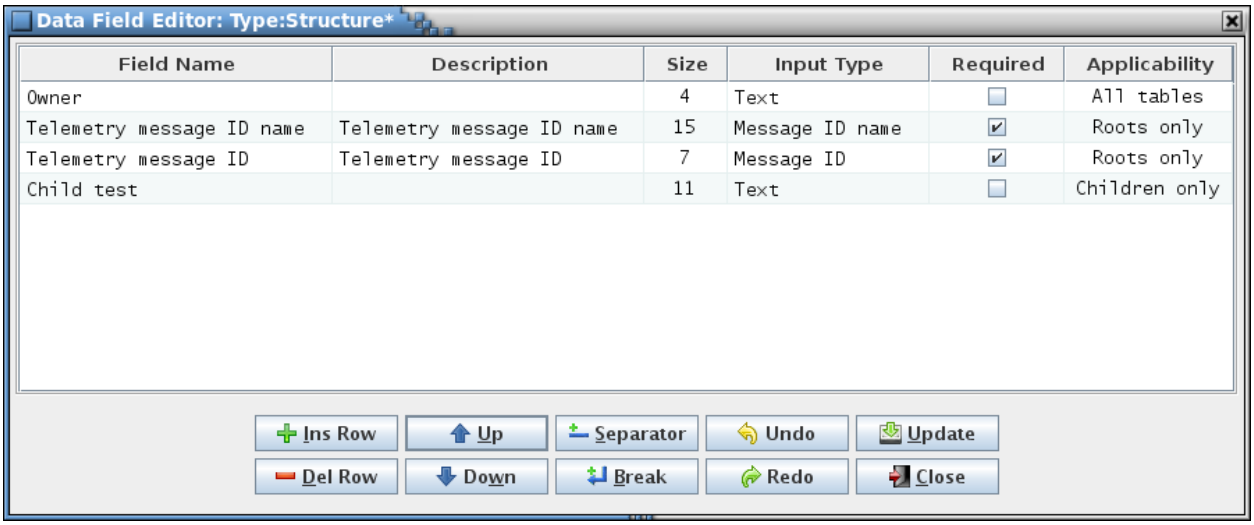


Figure 7. Data field editor

There are five or six columns in the field editor (depending on context):

Field Name This is the name of the data field. The name can be of any length and can contain letters, numerals, and punctuation characters. When the data field is displayed the field name is the label shown immediately to the left of the input field that is used to contain the field's value. The field name is also used if the data field is referenced from a script (see paragraph 4.10). The field name is required.

Description The field description is use to describe the content of the data field. The description appears as tool tip text whenever the mouse pointer hovers over the field in a table, group, or table type editor. HTML tags may be entered to provide formatting for the displayed tool tip text. This column may remain blank.

Size The field size defines the width, in characters, of the data field's input text field. Due to padding and font differences, the actual field width may appear slightly larger. The size must be a positive integer, and is required. If a boolean input type or one with selection items is used in the **Input Type** column then the size is ignored; a boolean field is

displayed as a check box, and the longest selection item determines the combo box width for a field with selection items.

Input Type The field input type constrains the type of value entered into the data field's input text field. If the value entered into the data field doesn't conform to field's specified input type then a warning message dialog is displayed and the field reverts to its previous value. The input types are selectable from the combo box pull-down menu that appears when a cell in the Input Type column is selected. See paragraph 4.7 for information on the available input types. A field with a boolean input type is displayed as a check box. The input type defaults to "text."

Required The **Required** column displays a check box, initially unselected. Selecting the check box indicates that the field is required. When the field is displayed in the table or table editor the input text field is highlighted in yellow as long as the field's input text field value is empty. The application does not enforce the user to input data into the fields marked "required"; the highlighting is used merely as a reminder that the field value is considered necessary and should be filled. A boolean (check box) data field does not display the highlighting.

Applicability This column only appears when assigning default data fields in the table type editor for a structure table type, or when editing the fields for a prototype or root structure table.

If the data field editor is invoked from the table type editor then it allows defining fields that are propagated to new and existing structure tables. Select the applicability for a field from the combo box pull down menu that appears when the applicability cell is selected. *All tables* indicates that the field will be added to all structure tables (root and child tables), *Roots only* implies that only root tables have the field added, and *Children only* means that only child tables have the field added. The default is *All tables*.

If the data field editor is invoked from a prototype or root structure table then all fields assigned to the table, regardless of the applicability setting, are displayed in the editor. For example, if the table has both *Roots only* and *Children only* fields assigned then only one of these types is displayed in the table's editor since a table can only be a root or a child, but not both. However, a table can change from a root to a child, and vice versa; when this happens the types of fields that are displayed are changed to match. The field editor displays all of the fields, whether applicable or not, so that these can be altered by the user.

The order that the rows appear in the field editor determines the order of appearance in the table editor, group manager, or table type editor. Field definition rows may be rearranged as desired by first selecting a cell in one or more rows, then pressing the **Up** or **Down** buttons to move the selected row relative to the ones not selected. The editor columns can be sorted as described in paragraph 4.4; however, the sorted order does not dictate the field display order.

Line separators and line breaks may be inserted as rows using the **Separator** and **Break** buttons respectively. Without these breaks the data fields, when displayed in a table, group manager, or table type editor, are arranged end to end, wrapping to the next line when the width of the editor is reached. The line break forces the next data field to the next row regardless of the editor width constraint. The line separator does the same, except that a dividing line is drawn between the rows where the separator is inserted. These breaks can be used to aid in grouping related data fields.

The field editor button commands are described below:

Ins Row The editor is initially empty unless the table editor, group manager, or table type editor from which it's invoked has any previously defined fields. To add a field first select the **Ins Row** button; a new field definition row is inserted into the editor. Additional rows can be added in the same manner. The insertion point is dependent on the currently selected row in the editor; if no row is selected then the new row is added at the bottom. The empty row has the Field Name and Size columns highlighted in yellow. The highlighting indicates that these columns are required and must have values assigned.

Del Row One or more field definition rows may be deleted by first selecting a cell in the target row(s), then pressing the **Del Row** button. The selection of multiple rows is constrained to contiguous rows; i.e., rows cannot be skipped.

Up The order that the rows appear in the field editor determines the order of appearance in the table editor, group manager, or table type editor. Field definition rows may be rearranged as desired by first selecting a cell in one or more rows, then pressing the **Up** button to move the selected row(s) up a row relative to the ones not selected.

Down Similar to the **Up** button action, except that selected row(s) are moved down a row relative to the ones not selected.

Separator Inserts a line separator below the currently selected field's row.

Break Inserts a line break below the currently selected field's row.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, redo, etc.).

Update Applies the data field definitions currently displayed in the field editor to the table editor, group manager, or table type editor from which the field editor was invoked. Any existing fields that are in the table editor, group manager, or table type editor are deleted and replaced by the new definitions. However, these changes are not stored in the database – this is only accomplished when the **Store** button in the table editor, group manager, or table type editor is selected.

Close Closes the field editor window. If any changes made have not been applied using the **Update** button then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.7 Input Types

Data table columns and data fields are assigned an input type in the table type editor (see paragraph 4.9.3.10) and the data field editor (see paragraph 4.6.1) respectively. The input type constrains the type of value entered into a data table cell or data field. Leading and trailing white space characters (spaces, tabs, etc.) are ignored and eliminated when the text is formatted (except for the two Text input types that specifically preserve these characters; spaces between characters in a text string are always retained). Leading plus (+) signs and zeroes are allowed for non-negative integer and floating point values, but are ignored and eliminated.

Some input types constrain the allowable values to one or more selection items (the **Variable reference** type, for example, and custom input types defined by the user that are provided a selection item list). The list items are displayed in a drop down menu in the table cell or data field. An empty cell/field is an allowable entry for input types with selection items and a blank item is automatically prepended to the list. These cells and fields use an item matching feature that allows for quickly pruning the selection item list based on the characters entered into the field or cell. As the user enters text those list items

not matching the text are removed from the list. Wildcard characters are allowed. A '?' matches a single character and a '*' matches one or more characters. The matching operation assumes a '*' is at the end of the entered text. Matching is case sensitive. For example, given the list below, if the match criteria entered is '*az' then the first four items constitute a match, whereas the remaining two items do not.

az
azq
a,az,b[0]
a,az,b[1]
b,Az
b,a,z

A blank item is always retained as the first item in the list. If editing ends (e.g., by selecting another field or cell) then the first item in the list that matches the current match text is entered into the field/cell. If no item in the list matches the match text then the field/cell is blanked. The match text can be used to reduce the number of items in the list, then the mouse or keyboard (i.e., arrow keys) used to select an item from the list.

Customized input types may be defined using the input type editor; see paragraph 4.9.3.12.

In the table type or data field editor select the row in the **Input Type** column corresponding to the data table column or data field. A combo box pull down menu appears with the input types, both default and custom, listed in alphabetical order. The default input types are:

Alphanumeric This type allows letters, numerals, and underscore characters. A numeral may not begin the text string. Alphanumeric text is appropriate for variable names.

Alphanumeric (multi) Allows multiple **Alphanumeric** inputs, separated by one or more white space character(s).

Argument name Special format used to designate a command table argument name column. This type has the same constraints as the **Alphanumeric** type.

Array index Special format used to designate the array size column. Allows one or more integer values (each greater than 1), separated by commas. For the array size column each value represents an array dimension size (e.g., if the array size is 2, 3, 4 then the associated array size is defined by *arrayName*[2][3][4]).

Bit length Special format used to designate the structure table bit length column. This type has the same constraints as the **Positive integer** type.

Boolean A check box is used to represent data table cells and data fields of this type.

Command code Special format used to designate the command table command code column. This type has the same constraints as the **Hexadecimal** type.

Command name Special format used to designate the command table command name column. This type has the same constraints as the **Alphanumeric** type.

Command reference Displays a drop down menu containing information for every command in the project (name, code, table, and number of arguments). Only a command from the list, or no selection, is allowed.

Description Special format used to designate a description column. This type has the same constraints as the **Text** type.

Enumeration Special format used to designate a column containing enumerated values. This type allows letters, numerals, and punctuation characters.

Floating point This type allows floating point values to be entered; i.e., values with decimal components in the form "#.###" (e.g., 3.14).

Hexadecimal This type allows only hexadecimal digits to be entered (0 – 9, A – F, and a – f). The hexadecimal digits may optionally be preceded by "0x".

Hexadecimal range Allows one or two **Hexadecimal** values. If two values are entered they must be separated by a hyphen (-).

Integer The integer data type allows input of any integer value: positive, negative, or zero.

Integer > 1 Only integer values greater than 1 are allowed for this type.

Maximum Special format used to designate a column containing maximum values. This type allows boolean, integer, floating point, and hexadecimal values depending on the data type associated with it. If the associated data type is missing or blank then the maximum value cell is blanked and cannot be edited. The maximum column is automatically paired with a minimum column (if present); if multiple minimum columns are present then pairing is done in order of column appearance in the table type definition. When paired the minimum value is constrained to be less than or equal to the maximum value.

Message name & ID Used to designate a table column or data field as representing a telemetry or command message name and ID. The contents must be in the format *[<message name>] <message ID>*. The field may be blank, contain only the message ID, or contain the message name and ID (examples: `GNC_NOOP_MID 0x1234` or `0xa001`). The constraints for the message name are identical to the **Alphanumeric** type. The constraints for the message ID are identical to the **Hexadecimal** type, except that the '#' character may be appended to the ID number. This character flags the message ID as being protected from overwriting by the automatic message ID assignment command. Assigning this type to a table column or data field enables the application to recognize the information as representing a message name and ID; see paragraph 4.9.3.14.3 for more information on how this input type is used.

Message reference This type causes a combo box pull down menu to appear when the table cell or data field is selected. The menu contains all of the currently defined message ID names, associated message ID numbers, and the message owners (table, group, or telemetry scheduler) in the format *<message name> (ID: <message ID>, owner: <owner name>*.

Minimum Special format used to designate a column containing minimum values. This type allows boolean, integer, floating point, and hexadecimal values depending on the data type associated with it. If the associated data type is missing or blank then the minimum value cell is blanked and cannot be edited. The minimum column is automatically paired with a maximum column (if present); if multiple maximum columns are present then pairing is done in

order of column appearance in the table type definition. When paired the minimum value is constrained to be less than or equal to the maximum value.

Negative float Similar to the **Floating point** type, except that only negative values are allowed.

Negative integer Similar to the **Integer** type, except that only negative integer values are allowed.

Non-negative float Similar to the **Floating point** type, except that only zero or negative values are allowed.

Non-negative integer Similar to the **Integer** type, except that only zero or negative values are allowed.

Non-zero hexadecimal Similar to the **Hexadecimal** type, except that a value of zero (0x0) is excluded.

Number Identical to the floating point type, except that a trailing '.0' isn't added for integer values.

Positive float Similar to the **Floating point** type, except that only positive, non-zero values are allowed.

Positive integer Similar to the **Integer** type, except that only positive, non-zero integers are allowed.

Primitive This type causes a combo box pull down menu to appear when the cell is selected. The menu contains all of the primitive data types. This selection is not available in the data field editor.

Primitive & Structure This type causes a combo box pull down menu to appear when the cell is selected. The menu contains all of the primitive data types along with the names of all the prototype structure tables. This is primarily for use in defining the **Data Type** column in structure tables but can be used elsewhere. This selection is not available in the data field editor.

Rate Special format used to designate telemetry sample rate columns. Allows positive integer values and values in the form "1 / #" where # is a positive integer value.

Text This type allows letters, numerals, and punctuation characters. **Text** is the default data type.

Text (multi-line) This type allows letters, numerals, punctuation, and new line characters, the latter allowing for multiple lines of text in a data field. The field initially displays as a single row, but the number of rows increases as new line characters are entered. This type behaves identically to **Text** if used as a table column's input type.

Text (spaces) Identical to the **Text** type except that leading and trailing white space characters are preserved.

Text (multi-line, spaces) Identical to the **Text (multi-line)** type except that leading and trailing white space characters are preserved. This type behaves identically to **Text (spaces)** if used as a table column's input type.

Units Special format used to designate a column containing units (e.g., degrees F, rpm, m/s). This type has the same constraints as the **Text** type.

Variable name — Special format used to designate a variable name column. This type has the same constraints as the **Alphanumeric** type.

Variable path — This is a special format valid only in a structure table type definition. If a column in the table is assigned this type then the table's path is combined with the variable name and data type to produce the variable's full path. The column can be edited, in which case the manually entered value overrides the automatically generated path. The characters that separate each structure reference in the path, and the data types and variable names, and whether or not the data type is displayed, are set using the **Show variables** command menu command; see paragraph 4.9.3.17.

Variable reference — Displays a drop down menu containing every structure and variable in the project. Only a structure or variable name from the list, or no selection, is allowed.

XML: Application ID — Name of the variable in the telemetry and command header structure tables that contains the application ID. This type has the same constraints as the **Alphanumeric** type. Used during XML import and export to determine the location from which to retrieve or where to store the application ID.

XML: Command Header — Name of the structure table representing the command header. This type has the same constraints as the **Alphanumeric** type. Used during XML import and export to indicate the command header table.

XML: Function Code — Name of the variable in the command header structure table that contains the command function code. This type has the same constraints as the **Alphanumeric** type. Used during XML import and export to determine the location from which to retrieve or where to store the command function code.

XML: Telemetry Header — Name of the structure table representing the telemetry header. This type has the same constraints as the **Alphanumeric** type. Used during XML import and export to indicate the telemetry header table.

4.8 Data Streams

CCDD supports the definition and use of multiple data streams. In this context a data stream refers to an uplink/downlink path; for example, serial, Ethernet, radio, etc. Each data stream has its own set of rate parameters (see paragraph 4.9.4.4). Addition of a stream is accomplished by adding a new sample rate column to a structure table definition (see paragraph 4.9.3.10 for information on altering a table type). A rate column is designated by assigning the column an input type of 'Rate' (see paragraph 4.7 for information on input types). A telemetry parameter can be assigned a rate in each of the defined data streams. The link manager (paragraph 4.9.4.1) allows linking telemetry parameters for allocation in the downlink messages. These linkages are specific to a data stream. The data stream can be assigned a name different from its associated rate column name in the rate parameter dialog (paragraph 4.9.4.4).

4.9 Command Menu

The following paragraphs describe the main application window's menu commands.

4.9.1 File

The **File** menu provides selections for connecting to the database, altering the database connection properties, reading, printing, and searching the application logs, enabling the web server, updating the application's overall appearance, and exiting the program.

4.9.1.1 Select user

When the **Select user** command is issued, if any table editor or the table type editor is open and has unsaved changes then a confirmation dialog first appears, allowing the user to choose whether to discard the unsaved changes and continue with the user change, or to cancel the user change. If there are no unsaved changes or if the user confirms discarding the changes then the editors are closed and the **Select User** dialog is displayed. The dialog allows entering the user name and/or user password. The appearance of the dialog is dependent on whether or not a connection is currently established with the PostgreSQL server. If no connection exists then the dialog appears as in Figure 8, and both the user and user password must be entered. If a connection to the server does exist (i.e., if changing to another user from one already connected to the server) then the dialog appears similar to that in Figure 9. For this case the user text field is replaced by radio buttons providing an alphabetized list of the users registered in the PostgreSQL server.

Select or type in a user name and, if required by the server, provide the password in the **Password** field, then select the **Okay** button. An attempt is then made to establish a connection as the indicated user with the most recently selected or open project's database. If a project's database is open when the user is changed and the newly selected user does not have access privileges to this project then the database is closed. Select the **Cancel** button to exit the dialog without changing the user.

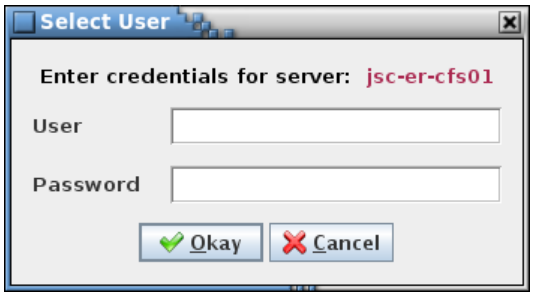


Figure 8. Select User dialog (no server connection)

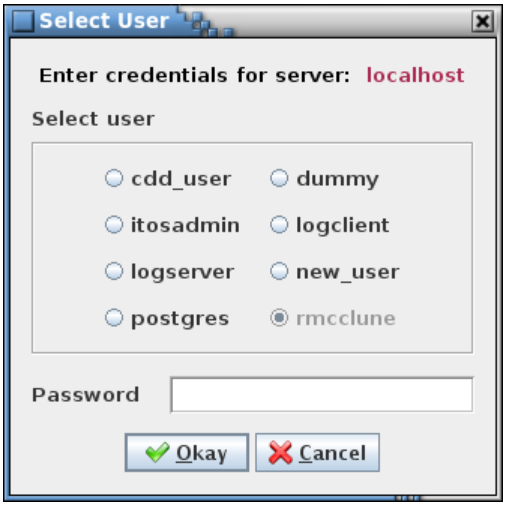


Figure 9. Select User dialog (server connected)

4.9.1.2 Database server

The **Database server** command is used to set the PostgreSQL database connection properties. When the command is issued, if any table editor or the table type editor is open and has unsaved changes then a confirmation dialog first appears, allowing the user to choose whether to discard the unsaved changes and continue with the server change, or to cancel the server change. If there are no unsaved changes or

if the user confirms discarding the changes then the editors are closed and the **Database Server** dialog is displayed (see Figure 10). The dialog allows entering the PostgreSQL server host name and server port number, and to enable or disable secure sockets layer (SSL) authentication.

Figure 10. Database server dialog

The dialog allows selection of a PostgreSQL server on the local or a remote host. Enter the server host name and the server port number, set the **Enable SSL** check box appropriate to the target connection's encryption requirements, and then select the **Okay** button. The host text field uses auto-completion to fill in the host string. The previous 30 host names are remembered by default, including those from previous sessions (the number of remembered host names can be changed via the Preferences dialog). If the host field is empty then the default host name, localhost, is used. The default port number for the PostgreSQL server is 5432. An attempt is then made to establish a connection to the server on the new host as the current user. Select the **Cancel** button to exit the dialog without changing the database server properties.

The SSL selection must match that of the target or else the connection will not be established. *Note: When SSL is enabled no certificate validation is performed by the JDBC driver.*

4.9.1.3 Read log

The **Read log** command causes the **Open Event Log** file selection dialog to be displayed. Navigate to the location of the desired CCDD event log file, highlight it using the mouse or keyboard controls, and select **Open** to open the log in a window similar to the main program window. The log file names are in the format CCDD-*YYYYMMDD-hhmmss*.log, where *YYYYMMDD* is the year, month, and day, and *hhmmss* is the hour, minute, and second when the log was created. Select **Cancel** to close the file selection dialog without opening a log file. See paragraph 4.2.5 for details on the event log columns and filter selections. The log window may be resized. The **Search** button displays the event log search dialog (see paragraph 4.9.1.5); the log can be searched for a user-specified text string. Select the **Print** button to open a printer selection dialog in order to print a copy of the log to the selected printer. Select **Close** to close the log window.

4.9.1.4 Print log

The **Print log** command causes a dialog to appear allowing selection of a printer in order to print a copy of the current session's event log; i.e., the log displayed in the main application window.

4.9.1.5 Search log

The **Search log** dialog provides a means of searching the current session's event log for a specified text string (see Figure 11). Case sensitivity for the search is governed by the **Ignore text case** check box.

The **Allow regular expression** check box, when checked, allows the use of a regular expression to define the search pattern in the search text field. A regular expression can be formulated to find multiple matching conditions (for example, the search for **a.c** would match any string that has a single character between the characters 'a' and 'c'). Information regarding the use of regular expressions is beyond the

scope of this document; however, resources and tutorials can be found online. When unchecked, the search text is matched as typed in the search text field.

Enter the search text in the input field and select the **Search** button. The search results are displayed in the table at the bottom of the search dialog. The first column, **Log Index**, shows the log entry's index number where a match is found. The second column, **Column Name**, provides the column where the match occurs in the event log. The last column, **Context**, displays the string from log entry containing the search text, with the search text highlighted. The full length of the log message text is searched (and displayed, if a match is found), even if the message is truncated in the event log due to length restrictions. Pressing the Ctrl-F key sequence while the main application window has the focus also displays the event log search dialog.

The search text field uses auto-completion to fill in the search string. The previous search strings (those for the event log, tables, and scripts) are remembered, including those from previous sessions. The number of remembered search strings can be changed via the Preferences dalog, and defaults to 30. Case sensitivity for auto-completion is based on the **Ignore text case** check box selection state.

The input text can be changed and the **Search** button pressed again to initiate another search of the log. The search results can be output to a file or printer by selecting the **Print** button. To exit the search dialog select the **Close** button.

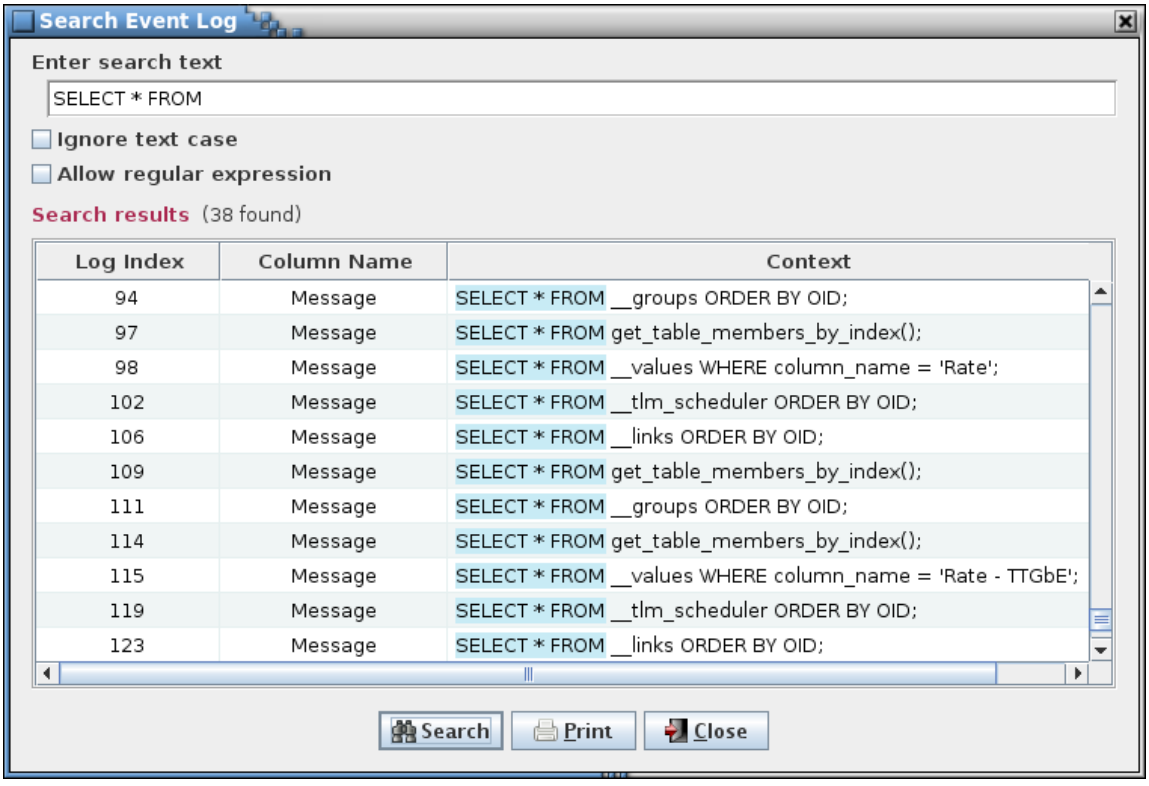


Figure 11. Search event log dialog

4.9.1.6 Web server

The embedded web server allows web-based applications access to a project's data. The web server must be started before any requests are made. If the application is running the **Enable server** command (paragraph 4.9.1.6.1) is used to start and stop the server. The server is disabled by default. Command line options are available to allow the server to be started at program start-up, with or without the GUI enabled. See Table 1 for the web server command line arguments.

All requests are directed to the currently open project database. The query format is:

host:*port*/*component*<?*attribute*<=*name*>>

The *host* name is the network name or IP address on which the CCDD application, with the web server active, is operating ('localhost' if active on the same machine as the requesting application). The *port* number is the port to which the server is assigned to listen (the default is 7070; this can be changed via command line command or menu option). The *component*, *attribute*, and *name* portions of the request determine the data returned. Data for tables, groups, applications, and the telemetry and application scheduler is available. Data may be requested for a single table, group, or application, or for all of the given component. Lists containing the names of all tables, groups, or applications can be requested. Table 6 contains the recognized *component*, *attribute*, and *name* combinations.

The data is returned to the requesting application in JSON "*key*":"*value*" pair format per the **Output** column in Table 6. For the initial request the user is prompted for a valid PostgreSQL server user name and password. Additionally, this user must have read access to the project open in the CCDD application hosting the web server.

If a table contains macro references then the table values default to replacing the macro names with the corresponding macro values, as defined in the macro editor (see paragraphs 4.5.7 and 4.9.3.12 for more information relating to macros). Requests can be made to return the table data with the macro names displayed in place of the macro values by appending **;macro** (or **;macros** – either is case insensitive) at the end of the *name* portion of the request, or at the end of the *attribute* portion if there is no *name* portion. For example, the request *table?=;macro* returns the table data for all tables with the macro names displayed, and the request *telemetry?;macros* returns the telemetry information for all tables with the macro names displayed in those JSON "*key*":"*value*" pairs that are taken directly from table cells.

Variable paths, formatted as single strings, can be requested as part of a structure table's data (see paragraph 4.9.3.17 for more information on variable paths). The variable paths are attached to the table data as an additional column named "Variable Path". The format for returning the paths is similar to the *variable* command (see Table 6):

;path<s><,<*variable path member separator character(s)*>,<*hide data types flag (*true *or* false*)*>,<*data type and variable name separator character(s)*>>

If the variable path member separator character(s), hide data type flag, and data type and variable name separator character(s) are not provided then the default values – "_", false, and "_", respectively – are used. The command is case insensitive and must be placed at the end of the *name* portion of the request, or at the end of the *attribute* portion if there is no *name* portion. It may be used in conjunction with the **macro** command described above. For example, the request:

table?data=;path,"_","false","."

would convert the variable path *rootStruct,parentStruct.structVar,int.primVar* to *rootStruct_parentStruct.structVar_int.primVar*, and the request:

table?data=;path,"_",true,"_"

would convert the variable path *rootStruct,parentStruct.structVar,int.primVar* to *rootStruct_structVar_primVar*.

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
table	all *(or blank)*	<*table name*>	Type, description, size (if a structure), data, and data fields for the specified data table (*table name* is case insensitive), or for all tables if *table name* is omitted	{"Table Name":"*table name*","Table Type":"*type*","Table Description":"*description*","Table Size":*size*,"Table Data":[{"*first row column name*":"*first* row *column value*",<,"*first row next column name:"first row next column value*{"*second row column name*":"*second* row *column value*",<,"*second row next column name:"second row next column value"*<,…>>}<,…>],"Data Field":[{"Field Name":"*field name*","Description":"*field description*","Size":"*field character length*","Input Type":"*input data type*","Required":"true *or* false","Applicability":"*field applicability*","Value":"*field value*"}<,*next field's data*…>]}
	data	<*table name*>	Data for the specified data table (*table name* is case insensitive), or for all tables if *table name* is omitted	{"Table Name":"*table name*","Table Data":[{"*first row column name*":"*first* row *column value*",<,"*first row next column name:"first row next column value*{"*second row column name*":"*second* row *column value*",<,"*second row next column name:"second row next column value"*<,…>>}<,…>]}
	description	<*table name*>	Description for the specified table (*table name* is case insensitive), or for all tables if *table name* is omitted	{"Table Name":"*table name*","Table Description":"*description*"}

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
	fields	<*table name*>	Data field information for the specified table (*table name* is case insensitive), or for all tables if *table name* is omitted	{"Table Name":"*table name*","Data Field":[{"Field Name":"*field name*","Description":"*field description*","Size":"*field character length*","Input Type":"*input data type*","Required":"true *or* false","Applicability":"*field applicability*","Value":"*field value*"}<,*next field's data*…>]}
	names	<*table type*>	Names of all tables of the specified table type (*table name* is case insensitive), or for all table types if *table type* is omitted	{"Table Type":"*table type*", "Table Names":["*table name*"<,"*next table name*"<,…>>]}
	size	<*table name*>	Size (in bytes) for the specified structure table (*table name* is case insensitive), or for all tables if *table name* is omitted	{"Table Name":"*table name*","Byte Size":*size*}
	search	<*search text*>,<*ignore case (*true *or* false*)*>, <*allow regular expression (*true *or* false*)*>,<*data table cells only (*true *or* false*)*><,*search table column names*>	Search for the specified text in the project database tables. Set the *ignore case* flag to true to ignore case when matching the text (defaults to false if not provided). Set the *allow regular expression* flag to allow a regular expression in the search text. Set the *data table cells only* flag to return only those matches within the data table cells (otherwise matches in the internal tables are returned as well; defaults to false if not provided). Optionally constrain the search to specific data table columns by appending a comma-separated list of column names (the names are case sensitive)	[{"Table / Object":"*table name*","Location":"*location within table*","Context":"*table cell contents containing the matching search text*"}<,…>]

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
proto_table	*Same requests as for* table *above, except only table information for prototype tables is returned*			
root_table	*Same requests as for* table *above, except only table information for root tables is returned*			
instance_table	*Same requests as for* table *above, except only table information for instance tables is returned*			
group	all *(or blank)*	<*group name*>	Description, associated table(s), and data field(s) for the specified group, or for all groups if *group name* is omitted	{"Group Name":"*group name*","Group Description":"*description*","Group Table":[“*table name*”<,”*next table name*<,…>>”],”Group Data Field”: [{"Field Name":”*field name*”,“Description”:”*field description*”,”Size”:”*field character length*”,“Input Type”:”*input data type*”,”Required”:”true *or* false”,”Applicability”:”*field applicability*”,“Value”:”*field value*”}<,*next field's data*…>]}
	tables	<*group name*>	Table(s) associated with the specified group (*group name* is case sensitive), or for all groups if *group name* is omitted	{"Group Name":"*group name*","Group Table":[“*table name*”<,”*next table name*<,…>>”]}
	description	<*group name*>	Description for the specified group (*group name* is case sensitive), or for all groups if *group name* is omitted	{"Group Name":”*group name*”,”Group Description”:”*group description*”}
	fields	<*group name*>	Data field information for the specified group (*group name* is case sensitive), or for all groups if *group name* is omitted	{"Group Name":"*group name*",”Group Data Field”: [{"Field Name":”*field name*”,“Description”:”*field description*”,”Size”:”*field character length*”,“Input Type”:”*input data type*”,”Required”:”true *or* false”,”Applicability”:”*field applicability*”,“Value”:”*field value*”}<,*next field's data*…>]}
	names		Names of all groups	{"Group Names":["*first group name*"<,"*second group name*"<,"…">>]}

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
application	*Same requests as for* group *above, except only group information for groups representing a CFS application is returned*			
scheduler	telemetry	<*data stream name*>, <*header size (bytes)*>, <*message ID name data field name*>, <*optimize flag (*true *or* false*)*>	Telemetry scheduler's copy table entries for the specified data stream	{"Stream Name":"*stream name*","Header Size":"*size*","Optimized":"true *or* false","Copy Table":[{"Input Message ID":"*input ID*","Input Offset":"*input byte offset*","Output Message ID":"*output ID*","Output Offset":"*output byte offset*","Number of Bytes":"*output size in bytes*","Root Table":"*root table name*","Variable Path":"*variable path*"} <,*next row*<,…>>]}
	application		Application scheduler's schedule table entries	*This command is currently ignored*
variable	<*variable path + name*> or blank	<*variable path member separator character(s)*>, <*hide data types flag (*true *or* false*)*>, <*data type and variable name separator character(s)*>	The variable path and name in the format used by the application paired with the converted path and name (dependent on the input parameters). If no variable path and name is supplied then every variable in the project along with its converted form is returned	{"*variable path and name (application format)*":"*variable path and name (user-specified format)*"[,…]}

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
telemetry	<<*group name*>, <*data stream name*>, <*sample rate*>>		Structure table name, variable name, data type, data stream information, enumeration information, and all other data associated with the variable for the telemetered variables in the structure tables belonging to the specified group (or application), or for all structure tables if *group name* is omitted. Filters can be specified for the data stream name and/or the sample rate so that only those telemetered variables matching the filter(s) are returned	{"Structure Table Name":"*table name from which the variable is taken*","Variable Name":"*variable name*","Data Type":"*variable's data type*","Data Streams":[{"Stream Name":"*first stream's name*","Sample Rate":"*variable's sample rate in this stream*"}<,*next stream's data*<,…>>],"Enumerations":[{"Enumeration Name":"first enumeration's name","Enumeration Value":"*first enumeration's value*"}<,*next enumeration's data*<,…>>]}
command	all *(or blank)*	<*group name*>	Command table name, command name, command code, command description, and for each of the command's arguments the argument name, data type, enumeration, minimum value, maximum value, and any other argument data, for the commands in the command tables belonging to the specified group (or application), or for all command tables if *group name* is omitted	{"Command Table Name":"*table name from which the command is taken*","*command name column name*":"*command name*", "*command code column name*":"*command code*",<"*command description column name*":"*command description*">,"Arguments":[{"*first argument's name column name*":"*first argument's name*","*first argument's data type column name*":"*first argument's data type*","*first argument's enumeration column name*":"*first argument's enumeration*","*first argument's minimum value column name*":"*first argument's minimum value*","*first argument's maximum value column name*":"*first argument's maximum value*"<,*first argument's other column data*>}<,*next argument's data*<,…>>]}

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
table_type			Table type definitions	{"Table Type Definition":[{"Column Name":,"*type name*"," Description":"*type description*","Input Type":"input data type","Unique":"true *or* false","Required":"true *or* false","Enable if Structure":"true *or* false","Enable if Pointer":"true *or* false"}]}
input_type			User-defined input type definitions	{"Input Type Definition":[{"Type Name":"*input type name*","Description":"*input type description*","RegEx Match":"*regular expression constraining the entered values*","Selection Items":"*list of text strings from which to select, separated by line feed characters*","Value Format":"*value format option:* Text*,* Array*,* Boolean*,* Float*,* Integer*, or* Hexadecimal"}<,*next input type definition*<,…>>]}
data_type			Data type definitions	{"Data Type Definition":[{"Type Name":"*user-defined name*","C Name":"*C-language name*","Size":"*size in bytes*","Base Type":"*base data type*"}<,*next data type definition*<,…>>]}
macro			Macro definitions	{"Macro Definition": ":[{"Macro Name":"*macro name*","Value":"*macro value*"}<,*next macro definition*<,…>>]}
message_id			Message ID owners, names, and values	{"Message ID Owner, Name, and Value": [{"Owner":"*message ID owner*","Message ID Name":"*message ID name*","Message ID":"*message ID value*"}<,*next message ID*<,…>>]}

Request			Returned Information	Output[1,2]
Component	**Attribute**	**Name**		
project_info			Get the active project's information (name, database name, description, lock status, current user name, project owner name, and project data fields)	{"Project":"*project name*","Database":"*database name*","Description":"*project description*","Status":"locked *or* unlocked","User":"*user name*","Owner":"*owner name*","Project Data Field":[{"Field Name":"*field name*","Description":"*field description*","Size":"*field character length*","Input Type":"*input data type*","Required":"true *or* false","Applicability":"*field applicability*","Value":"*field value*"}<,*next field's data*…>]}
authenticate	*user name*	*user password*	Determine if the specified user credentials are valid for the currently open project database	true *if the credentials are valid;* false *otherwise*
shutdown			Close the web server and project database, and exit the CCDD application	

1 *Only those JSON "*key*":"*value*" pairs that are members of a JSON array (i.e., that are enclosed by brackets ([]) in the output) have their original order preserved; other pairs may appear in any order.*

2 *If* Name *is omitted in a* table*,* proto_table*,* root_table*,* instance_table*,* group*, or* application *request then a JSON array is returned in the format* [<*first output*><*,second output*<,…>>]*, with each array member representing a table, group, or application.*

Table 6. Web data access commands

4.9.1.6.1 Enable server

Selecting the **Enable server** command toggles between starting and stopping the web server. When enabled, the web server allows CCDD to respond to web-based queries. When disabled, the server ignores web queries.

4.9.1.6.2 Select port

The **Select port** command displays a dialog (Figure 12) that allows selection of a port number for the embedded web server. Enter the server port number, then select the **Okay** button. If the web server is active then it's automatically restarted using the new port number. Select the **Cancel** button to exit the dialog without changing the server port.

Figure 12. Web Server dialog

4.9.1.7 Preferences

The **Preferences** command displays the **Preferences** dialog (Figure 13), which allows altering the application's display characteristics. This includes choosing the "look and feel" (L&F), fonts, colors, sizes, and spacing values applied to the program's GUI components. The dialog has a separate tab for each of the selection types. Once updates to the preferences are completed press the **Close** button to exit the dialog. The preference changes are saved so that the updated characteristics are used when the application is opened again. See Appendix E.2 for details on the stored preferences.

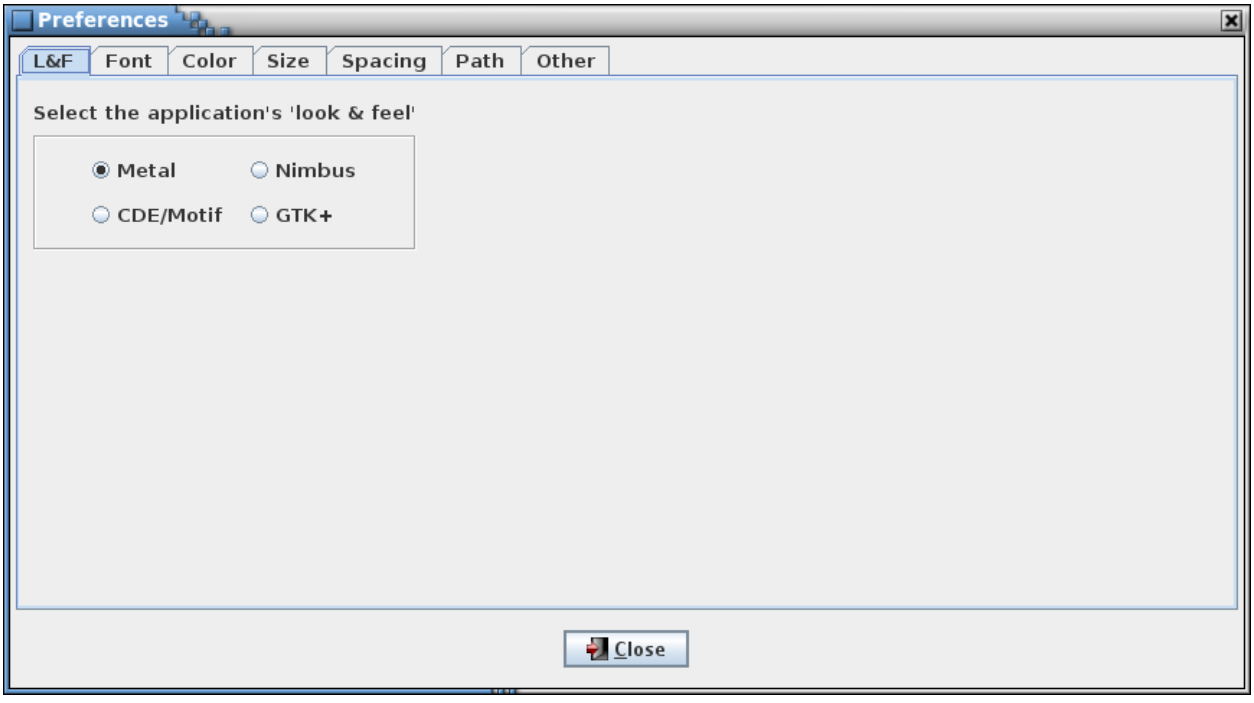


Figure 13. Preferences dialog; look and feel preferences

4.9.1.7.1 L&F

The L&F can be changed via the **L&F** tab (Figure 13). L&Fs change the shape and color scheme of the graphical components (see Figure 14), though the basic layout remains the same. The default is

"Metal", the standard L&F provided with Java. The list of L&F selections displayed in the dialog is dependent on the available L&Fs loaded on the host machine. When the radio button associated with the desired L&F is selected the **Preferences** dialog, main application window, and any other open CCDD windows are immediately redrawn to reflect the L&F chosen.

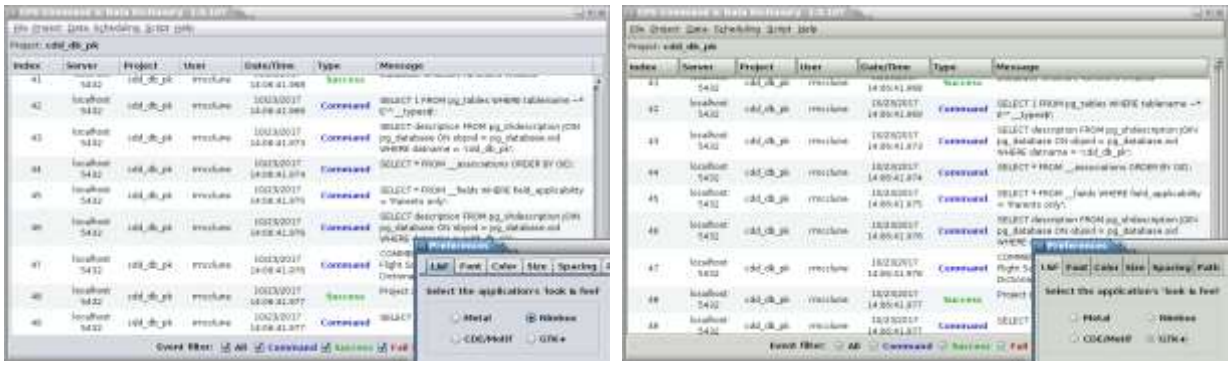


Figure 14. Example look and feel differences

4.9.1.7.2 Font

The **Font** tab displays the selection controls for the various fonts used in the application (Figure 15).

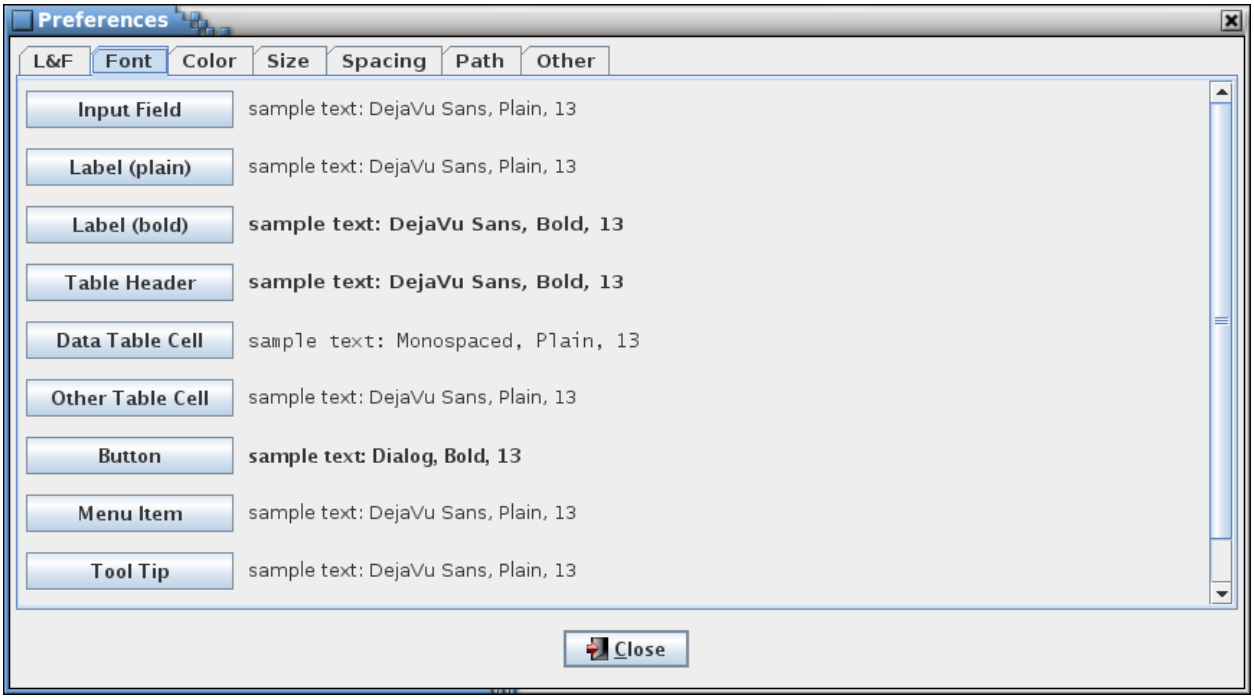


Figure 15. Font preferences

Each font is represented by a button with a sample of the font's text beside it. Hovering the mouse pointer over the font button displays a tool tip describing the font's use in more detail. To alter a font press its associated button – a font selection dialog appears (Figure 16). The font, style, and size are selected from the lists; the Sample field displays a sample of text using the chosen font settings. Select **Update** to change the selected font. The font change is implemented immediately in all of the application's open windows. Select **Default** to use the font's default settings. Press the **Close** button to exit the font selection dialog.

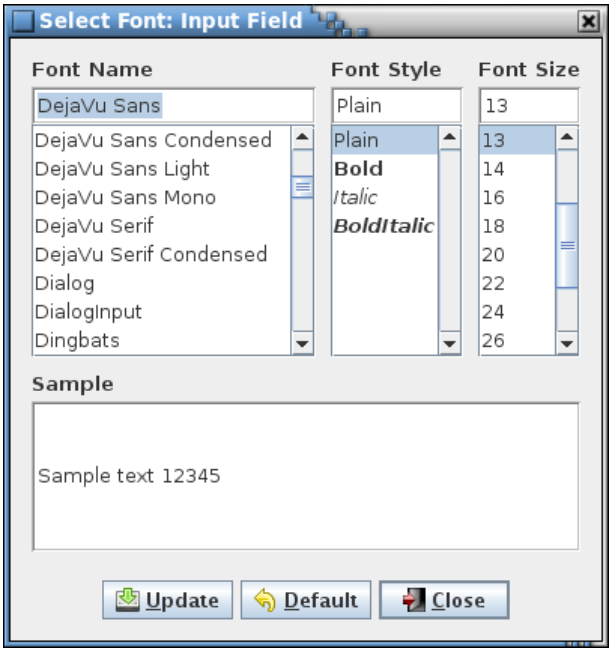


Figure 16. Font selection dialog

4.9.1.7.3 Color

The **Color** tab displays the selection controls for the various colors used in the application (Figure 17).

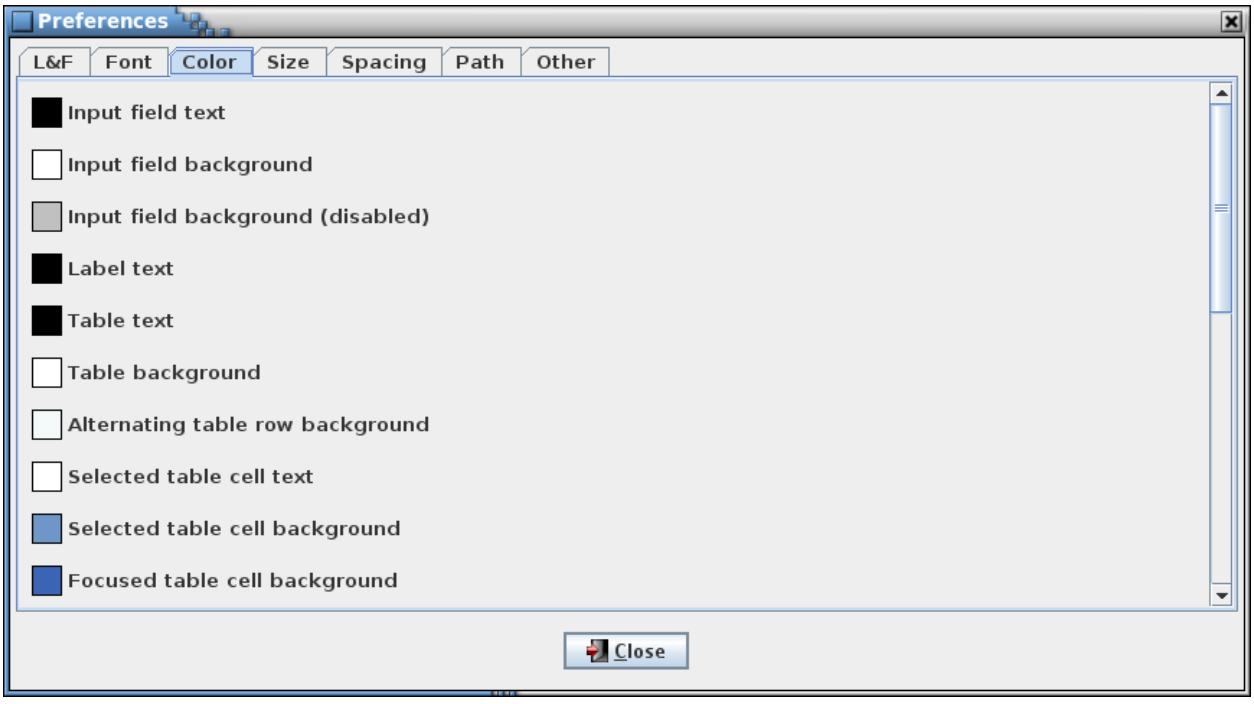


Figure 17. Color preferences

Each color is represented by a color box with a description beside it. Hovering the mouse pointer over the color box or description displays a tool tip describing the color's use in more detail. To alter a color select its associated box – a color selection dialog appears (Figure 18). The actual appearance of the color selection dialog varies based on the chosen L&F. However, the basic color selection for each is the same – use the mouse to operation alter the color parameters via the dialog's buttons, sliders, wheels, input fields, etc. Select **Update** to change the selected color. The color change is implemented

immediately in all of the application's open windows. Select **Default** to use the color's default settings. Press the **Close** button to exit the color selection dialog.

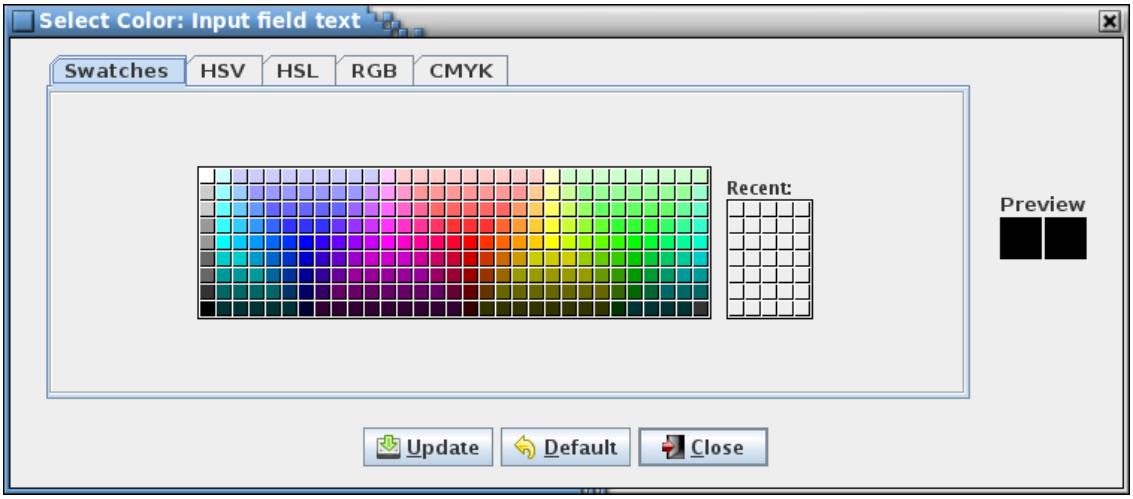


Figure 18. Color selection dialog

4.9.1.7.4 Size

The **Size** tab displays the selection controls for the various size values used in the application (Figure 19).

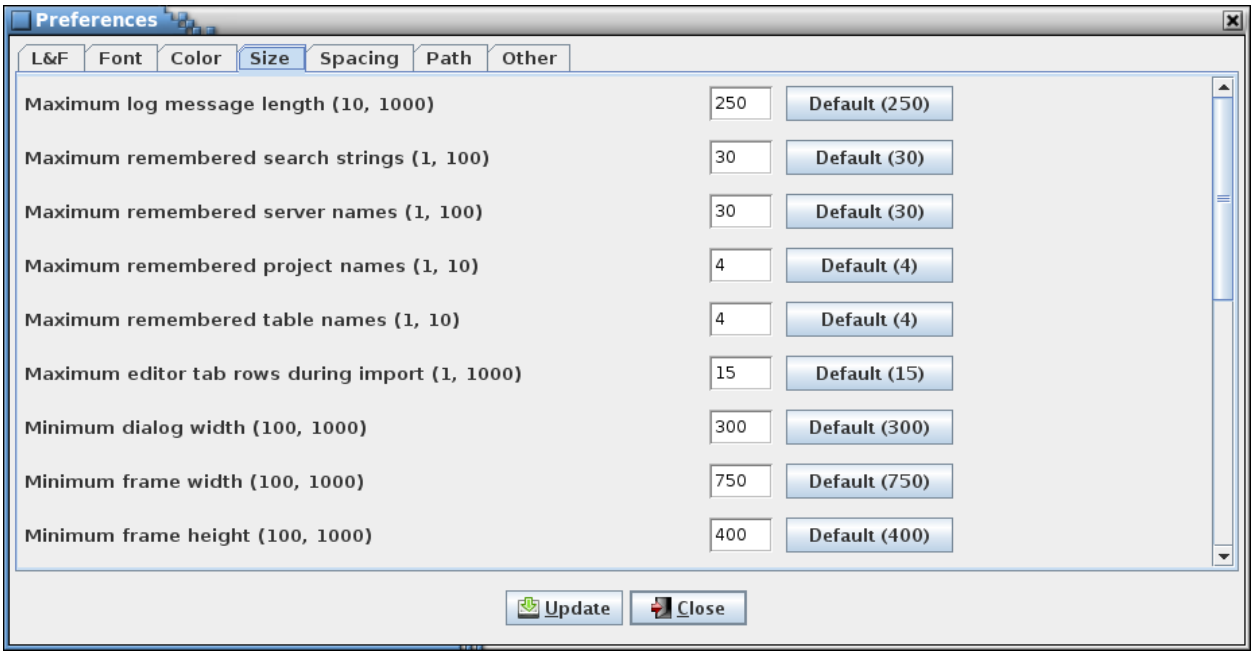


Figure 19. Size preferences

Each size value is represented by an input field with a description and default value button on either side of it. Hovering the mouse pointer over the description or input field displays a tool tip describing the size value's use in more detail. The description includes the minimum and maximum values allowed. To alter a size value enter a valid value into its associated input field. Selecting the size value's associated **Default** button (which displays the default value) enters the size value's default value into the input field. Select **Update** to implement the altered size value(s). Although size changes are implemented immediately, these are not apparent until the size value is used (e.g., the log message length applies to event messages subsequently logged).

4.9.1.7.5 Spacing

The **Spacing** tab displays the selection controls for the various spacing values used in the application (Figure 21).

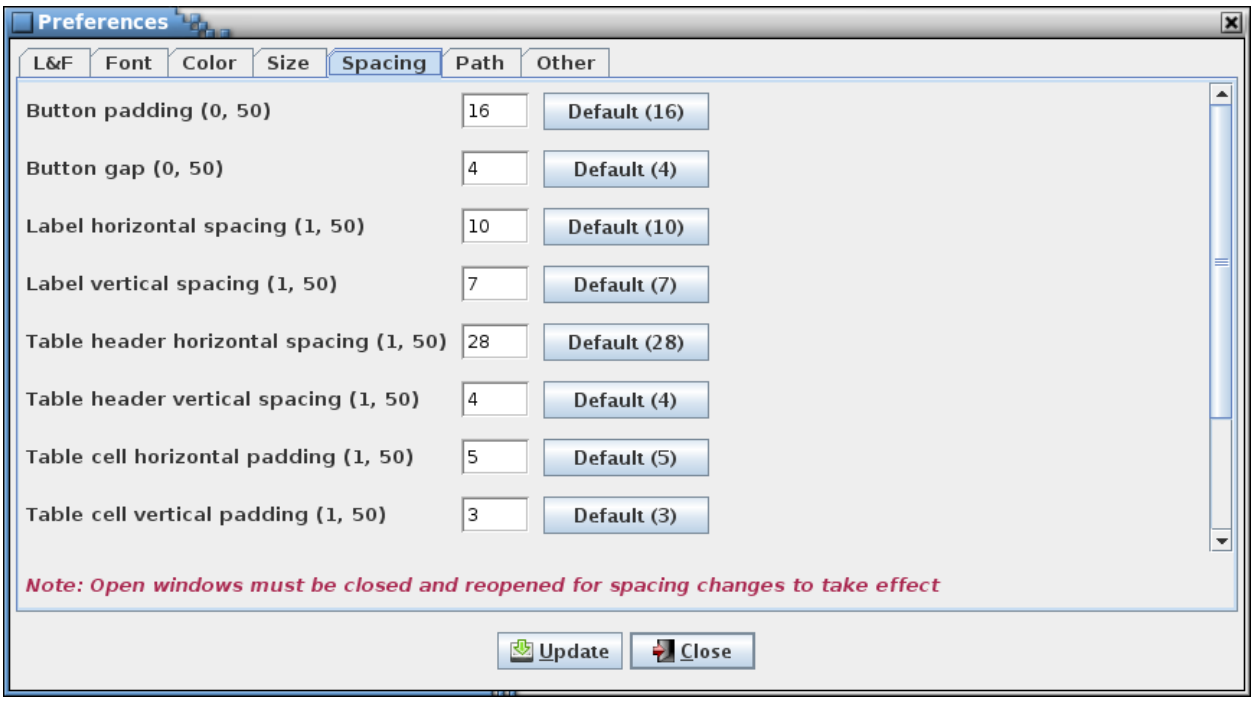


Figure 20. Spacing preferences

Each spacing value is represented by an input field with a description and default value button on either side of it. Hovering the mouse pointer over the description or input field displays a tool tip describing the spacing value's use in more detail. The description includes the minimum and maximum values allowed. To alter a spacing value enter a valid value into its associated input field. Selecting the spacing value's associated **Default** button (which displays the default value) enters the spacing value's default value into the input field. Select **Update** to implement the altered spacing value(s). As indicated by the note at the bottom of the dialog changes to spacing values are not applied to the application's open windows. These must be closed and reopened to use the updated values. Since this includes the main window, the application must be restarted in order for the spacing values to affect it.

4.9.1.7.6 Path

The **Path** tab displays the selection controls for the various folder paths used in the application (Figure 21).

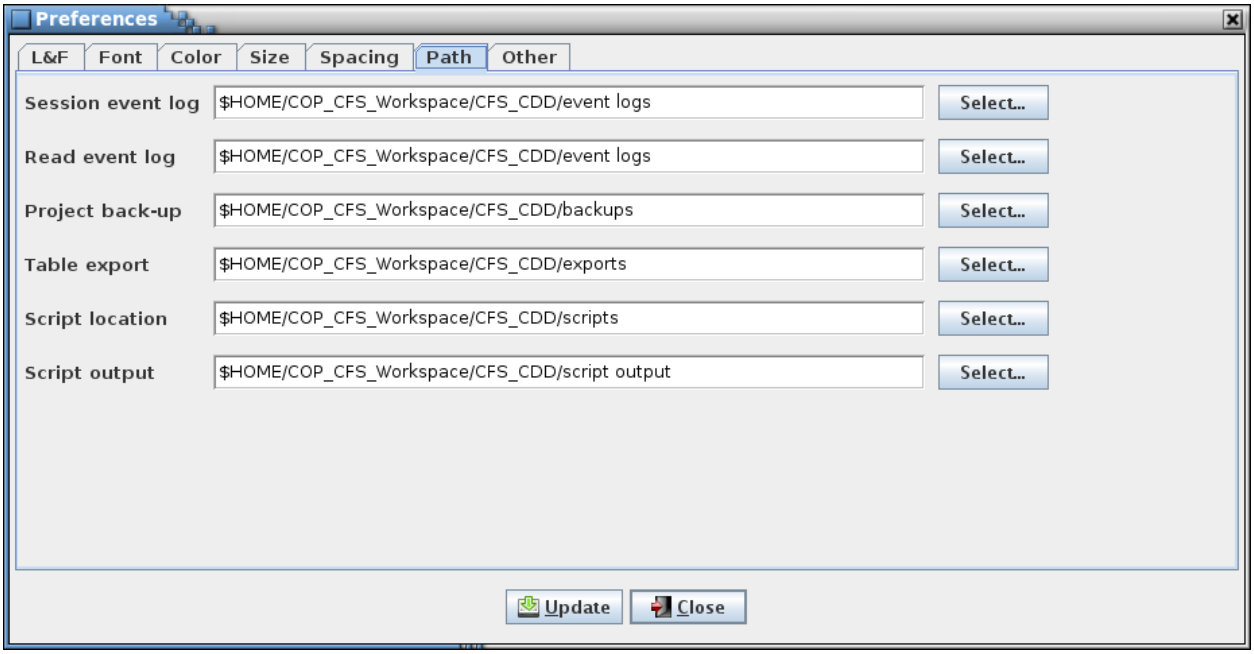


Figure 21. Path preferences

Each path is represented by an input field with a description and path selection button on either side of it. Hovering the mouse pointer over the description or input field displays a tool tip describing the path's use in more detail. To alter a path either enter it into its associated input field or press its **Select…** button to display a folder chooser dialog; once selected in the dialog the path appears in the input field. Select **Update** to implement the altered path(s). Paths can be absolute or relative. Relative paths are based on the folder in which the application is executed. Note that path selection in other dialogs (for example, the **Script Manager** or **Open Event Log** dialogs) updates the respective path if changed in the dialog. Path selection via command line argument also updates the path preference.

The **Script output** path is a special case. It does not directly affect the output location of files generated by scripts. Instead, a script may obtain the **Script output** path via an access method (see paragraph 4.10.7) and use it to set the location of any output files(s).

4.9.1.7.7 Other

The **Other** tab displays program preferences that don't conform to those in the other tabs (Figure 22).

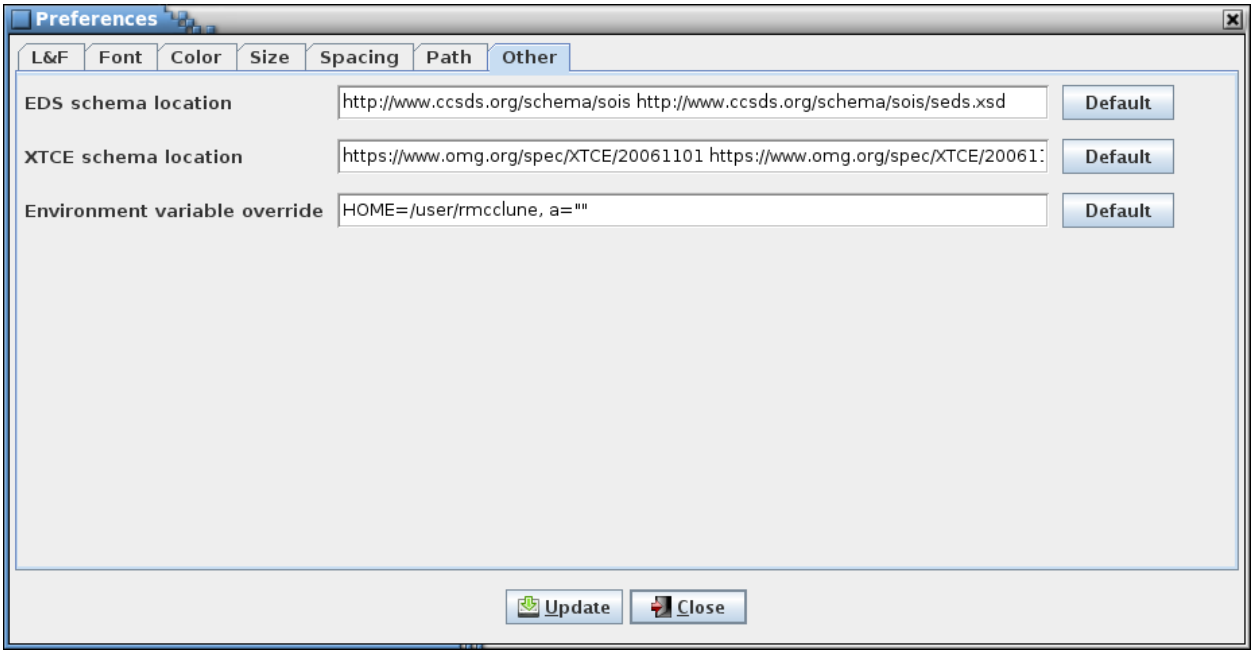


Figure 22. Other settings

Each setting is represented by an input field with a description and default selection button on either side of it. Hovering the mouse pointer over the description or input field displays a tool tip describing the setting's use in more detail. To alter a setting value enter a valid value into its associated input field. Selecting the setting's associated **Default** button enters the setting's default value into the input field. Select **Update** to store the altered setting value(s).

4.9.1.8 Exit

Choosing the **Exit** command displays a dialog so that the user can confirm whether or not to exit the application. Select **Okay** to exit CCDD. If there are unsaved changes to a table editor or the table type editor then the user is queried whether or not to continue. If **Okay** is selected the open editors are closed (any unsaved changes are discarded), the main application window is closed, and the application exits. Select **Cancel** to close the dialog without exiting the application.

4.9.2 Project

The **Project** menu contains commands for interacting with the project databases.

Each project has a locked/unlocked status flag. This flag is checked by the application when attempting to access a project. Project access is required for the **Open**, **New**, **Rename**, **Copy**, and **Delete** commands described in this section. If the flag indicates the project is unlocked the command proceeds. If the flag indicates the project is locked, the project access is denied and the specified operation is terminated. Access failure results in display of a database error dialog and the failure is written to the event log. The lock status is set to "locked" for an open project database. When the project database is closed (e.g., when exiting the CCDD application) the flag is set to "unlocked". Abnormal termination of the CCDD application can result in a project database retaining a locked status. The **Unlock** command (paragraph 4.9.2.9) allows clearing a project's lock status.

Note: In the project dialogs below, only those project databases for which the current user is allowed access are displayed.

4.9.2.1 Open

Selecting the **Open** command results in a dialog being displayed that shows the CCDD project databases, along with their descriptions, that are available in the PostgreSQL server (see Figure 23). The currently open project database is shown selected and grayed out. Other projects that are open in another instance of the CCDD application are also grayed out and have their associated radio button disabled. Select a project's radio button and then the **Open** button to open the selected project's database. The currently open project is first closed, along with any open table or table type editors. If the editors have any unsaved changes then a confirmation dialog appears, allowing the user to choose whether to continue with the project change, discarding the unsaved changes, or to cancel the project change. Select **Cancel** to allow the currently opened project to remain open.

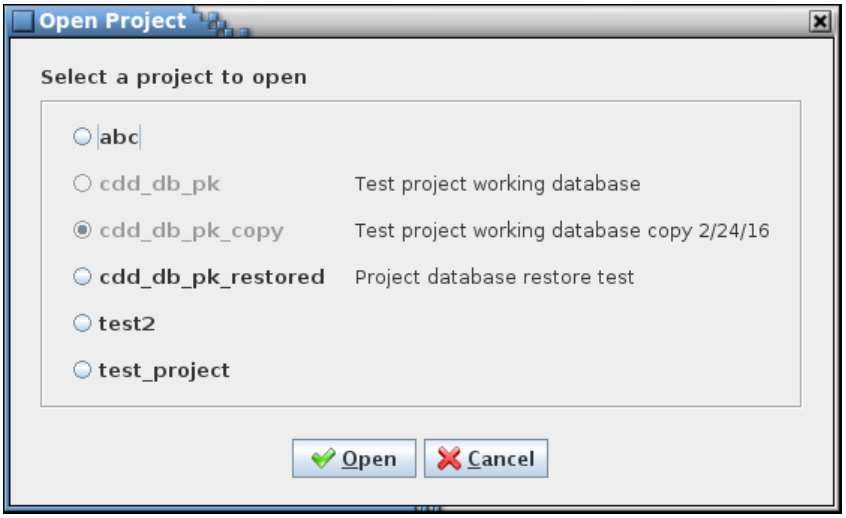


Figure 23. Select Project dialog

4.9.2.2 Close

When the **Close** command is selected the currently open project database is closed, along with any open data table or table type editors. If the editors have any unsaved changes then a confirmation dialog appears, allowing the user to choose whether to discard the unsaved changes and continue with closing the project, or to cancel closing the project.

4.9.2.3 New

The **Create Project** dialog (see Figure 24) appears when the **New** command is chosen, which allows creation of a new CCDD project database. A project owner must be selected from the list of available roles stored in the server, and a name supplied for the new project. Optionally, a description can be entered for the project.

The choice of owner should take into account the number of users that require access to the project's database. If only a single user needs access then that user can be selected as the owner. If multiple users need access then a group role should be created and this role assigned as the owner. All users requiring access would then need to be made members of this group role. Note that any user with super user status can access the project's database regardless of the owner. See paragraph **Error! Reference source not found.** for further information regarding setup of the PostgreSQL server.

The project name can contain any character except a semi-colon (;). The project name is used to create the project's PostgreSQL database name. To meet the PostgreSQL naming constraints all letters are changed to lower case, any non-alphanumeric characters are converted to underscores, and an underscore is prepended to the resulting name if it begins with a numeral. If the resulting database

name length exceeds 63 characters (the maximum allowed by PostgreSQL) then the name is truncated to the maximum length. The project and derived database names must be unique; these may not be the same as another project/database name existing in the server. The description can contain any character and, optionally, can be formatted using HTML tags.

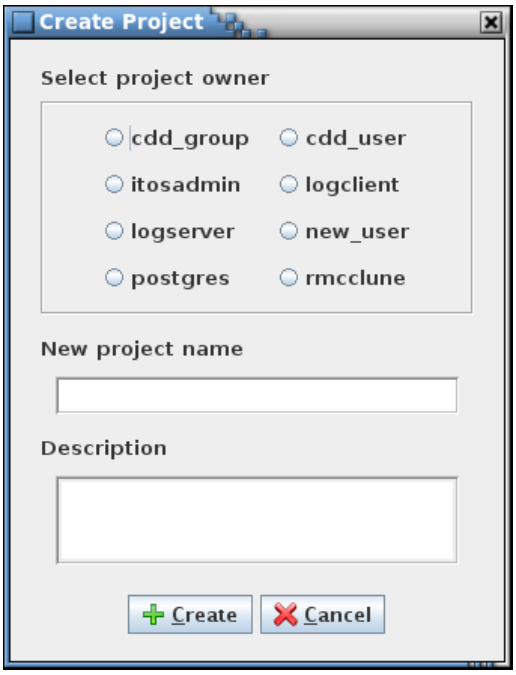


Figure 24. Create Project dialog

4.9.2.4 Rename

When the **Rename** command is selected the **Rename Project** dialog is displayed (see Figure 25). This dialog allows an existing project to be renamed, its description to be altered, or both. When one of the radio buttons representing a project's database is selected the name and description appear in the fields below the radio button panel. Projects that are open in another instance of the CCDD application cannot be renamed and are grayed out with their radio button disabled. Only a project's administrator may rename a project, so radio buttons for projects for which the user is not an administrator are also disabled. See paragraph 4.9.2.3 for constraints on the project name and description. When the **Rename** button is selected the project and description are updated. If the currently open project is renamed any open table editors are closed; if an editor has unstored changes then a dialog first appears allowing the user to confirm discarding the changes or canceling the rename operation. Note that this dialog can, if desired, be used to alter only the project's description (the description can also be edited using the **Manage project fields** command in the **Data** menu; see paragraph 4.9.3.15). Selecting the **Cancel** button closes the dialog without making any alterations.

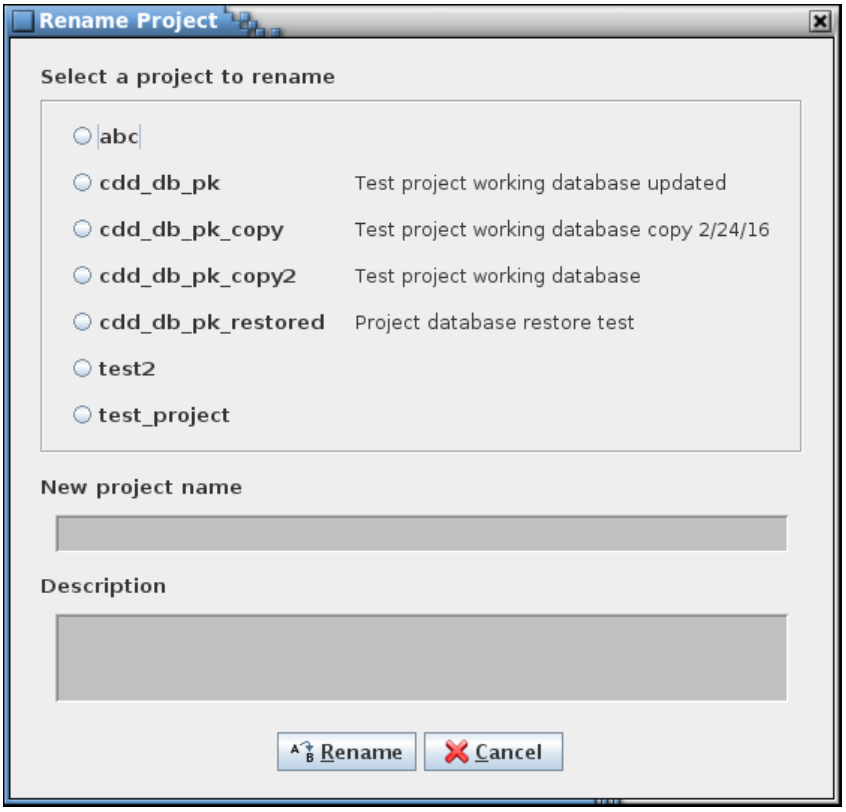


Figure 25. Rename Project dialog

4.9.2.5 Copy

When the **Copy** command selected the **Copy Project** dialog is displayed (see Figure 26). This dialog allows an existing project's database to be copied. Only a project's administrator may copy a project – the radio buttons for those projects which the user does not have administrator access are disabled. The exception is that a user with read/write access may copy the currently open project. When one of the radio buttons representing a project is selected the name and description appear in the fields below the radio button panel. The project name has the text "_copy" automatically appended, though the copy's name and description can be altered as desired. See paragraph 4.9.2.3 for constraints on the project name and description. The **Append date and time to project name** check box, if selected, replaces the text "_copy" with the current date (year, month, and day) and time (hours, minutes, and seconds) stamp. Deselecting the check box removes the data and time stamp and appends the text "_copy" again. When the **Copy** button is selected the selected project's database is copied, using the copy name and description. Selecting the **Cancel** button closes the dialog without making a copy.

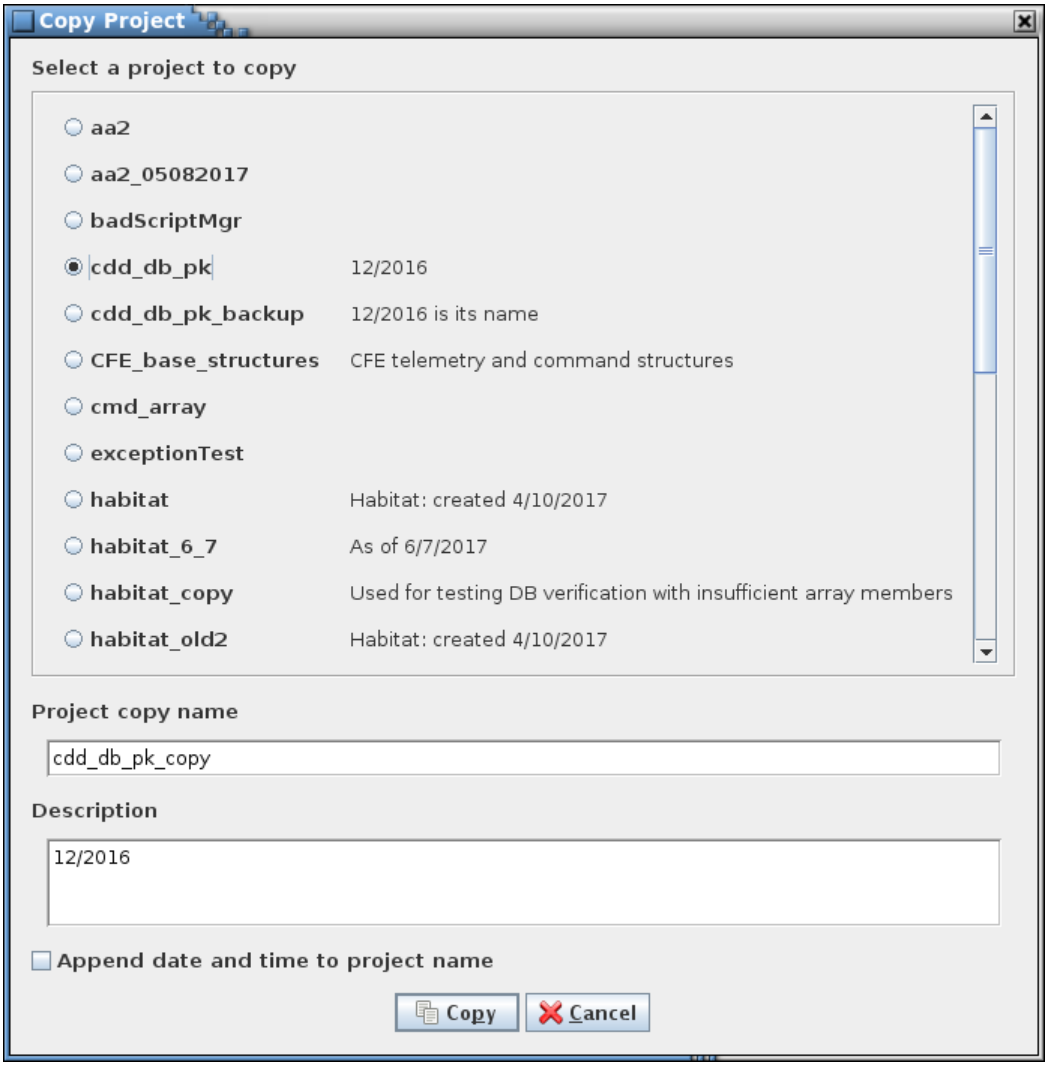


Figure 26. Copy Project dialog

4.9.2.6 Delete

The **Delete** command allows one or more project databases to be deleted. The **Delete Project(s)** dialog (Figure 27) appears when the command is issued. Projects that are open, in this or another instance of the CCDD application, cannot be deleted and are grayed out with their associated radio button disabled. Only a project's administrator may delete a project, so radio buttons for projects for which the user is not an administrator are also disabled. After selecting a project (or projects) to delete, selecting the **Delete** button removes the project database(s) from the server. Selecting the **Cancel** button exits the dialog without deleting any projects. If **Delete** is selected a confirmation dialog is displayed for each selected project; selecting **Okay** continues with the delete operation for that project, and **Cancel** ignores the indicated project and does not delete it.

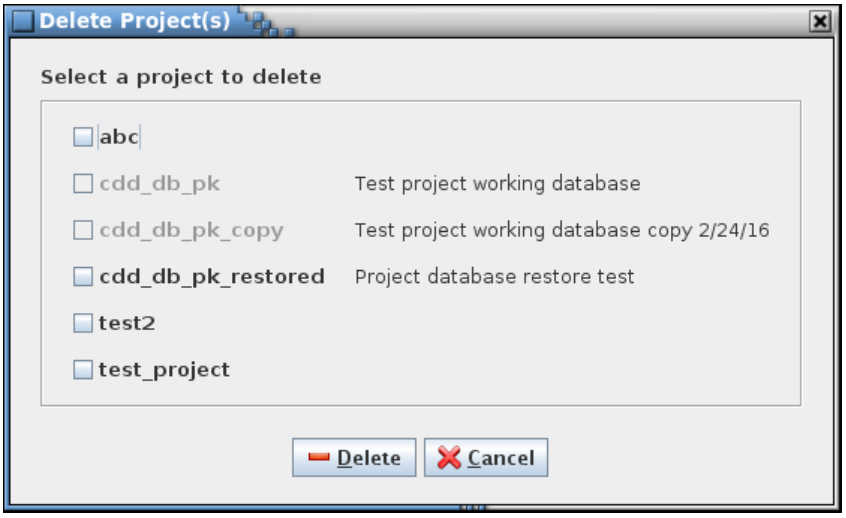


Figure 27. Delete Project dialog

4.9.2.7 Backup

The **Backup** command allows a user with read/write or administrator access to create a backup of the currently open project's database. A file selection dialog is displayed for choosing the location and name of the backup file (Figure 28). The backup file extension is '.dbu'. The **Append date and time to file name** check box, if selected, appends the current date (year, month, and day) and time (hours, minutes, and seconds) stamp to the file name. Deselecting the check box removes the data and time stamp. Select the **Backup** button to proceed; if the file selected already exists an overwrite confirmation dialog appears. The backup file is created using the PostgreSQL *pg_dump* command. This produces a PostgreSQL script file, in plain ASCII text, that has all of the commands necessary to create the project's database as it currently exists. The backup file makes it easy to transfer the database between servers and platforms. The **Restore** command, detailed in paragraph 4.9.2.8, uses the file generated by the **Backup** command to recreate a project's database. Selecting the **Cancel** button exits the dialog without creating a backup.

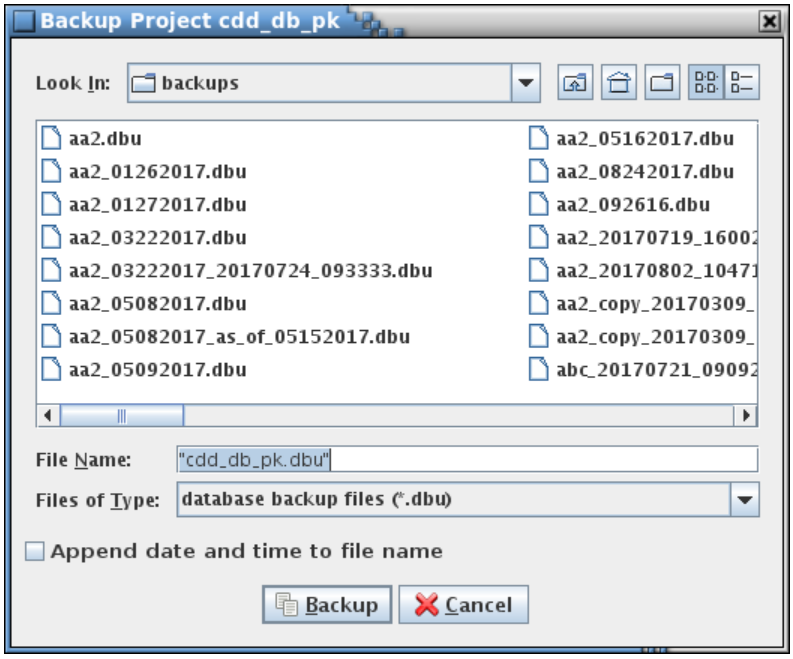


Figure 28. Backup Project dialog

4.9.2.8 Restore

The **Restore** command allows (re)creating a project's database on a server using a PostgreSQL script file created using the **Backup** command (see paragraph 4.9.2.7). Using the dialog that appears, navigate to the desired backup file, select it, and press the **Restore** button; the project database recorded in the script file is restored to the server. The name of the restored database is its original name with "_restored" appended, and the owner is set to its original owner.

If the name of the restored project's database would match that of an existing database then a sequence number is appended to the restored database's name. For example, if the database *abc* is restored and the database *abc_restore* already exists then the database is restored as *abc_restore1*; if *abc_restore1* already exists then *abc_restore2* is used, and so on until an unused name is found.

4.9.2.9 Unlock

The **Unlock** command allows the locked status to be changed to "unlock" for a project database. This command is intended to be used to remove a lock from a project that remains locked after abnormal termination of the CCDD application. The **Unlock Project(s)** dialog (Figure 29) appears when the command is issued. Though all projects are displayed, only those that are locked have their associated check box enabled.

The locked/unlocked status is displayed beside the project database name along with the name(s) of the user(s) that have active connections to the project. A project is shown as "Current" if opened by the current instance of the CCDD application and is shown as "in use by" the current user. "Locked" is displayed if a project is in use by another instance of the CCDD application or was open when the application terminated abnormally. "Unlocked" indicates that no other instance of the CCDD application has the project open. Other applications may have active connections to the project (e.g., the PostgreSQL command line interface application, psql). The users for these non-CCDD connections are also shown in the "in use by" list. Referring to Figure 29, project *abc* is unlocked with no active connections, project *cdd_db_pk* is open by user *new_user* in this instance of the CCDD application, project *test2* is not open by CCDD but does have an active connection from another application by user *rmcclune*, and project *test_project* is open by user *new_user* in another instance of CCDD.

After selecting a project (or projects) to unlock, selecting the **Unlock** button unlocks the project database(s). Selecting the **Cancel** button exits the dialog without altering the project lock statuses.

***Warning**: Removing a project's lock allows concurrent access to the project from more than one instance of the CCDD application. This may produce unexpected results or corruption of the project database if the multiple instances make updates to the update.*

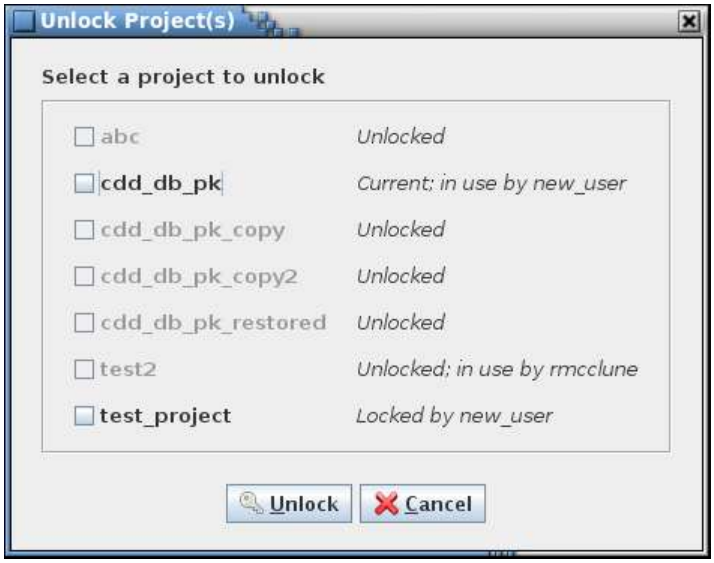


Figure 29. Unlock Project(s) dialog

4.9.2.10 Verify

The **Verify** command performs a consistency check on the currently open project database. This check ensures that the project's data tables are consistent with the table type definitions and that the information within a table is valid. Errors in the tables should not arise from interactions with the CCDD application. However, changes to the project's database from another application (e.g., psql) or using a version of the CCDD application that differs from the one used to create the project could result in the introduction and flagging of errors. The user is alerted to any potential problems and, where possible, is given the option to make corrections to the project's tables, ignore the problem and continue the check, or to cancel the check. There are three areas of verification performed, described in the following paragraphs. No changes are made to the project database until the user selects and confirms applying the updates at the end of the check. Since the project database can be altered by the verification it is recommended that the project be backed up or copied prior to allowing any updates to the database.

The *internal table check* verifies the project database's internal tables. These tables are for use by the CCDD application and are not directly viewable or editable by the user from within the application. The verification checks that the tables contain the expected number of columns and that the columns have the expected names and data types. Extraneous internal tables – tables with names conforming to the internal table naming scheme – are also detected; these tables can be created by the application's automatic update feature and can be ignored.

The *path reference check* verifies that the table and variable references in the internal tables are valid. Many of the internal tables store references to tables, and in the case of structure tables the table reference can include one or more child tables and associated variable names. When changes made to a table the references in the internal tables are updated as well. If any table or variable references no longer exist these references can be removed upon confirmation by the user.

The *table type check* compares each data table to its table type definition, verifying that the number of columns, column names, and column order match. Each data table is checked to see if its type matches one of the defined table types, and if so, that it contains only those columns defined for that type. If updating is confirmed then missing columns are added (devoid of data) and extra columns are eliminated (including the data in them). If the table's type is not defined then the entire table is deleted, along with its contents, when the update is applied. If the internal data type of the column doesn't match the expected one then the column data type is corrected with no loss of the column's data.

The *table data check* performs a check of the data within each table. In doing so it opens and inspects each data table and can generate a considerable number of database queries as is evidenced in the event log. Depending on the number of tables and the amount of data within them this operation can take a while. Each data value is checked to ensure it isn't null (empty cells in a data table contain blanks instead of nulls) and that it is compatible with the input type as defined in the table's type definition (e.g., no alphabetical characters in an integer cell). A check is made that each row in the table contains a row index (these indices are hidden from display in the table editor), that the row indices begin at 1, and there are no gaps in the index values. For structure tables containing array variables the check looks for missing array definitions (i.e., an array member without a corresponding array definition) and missing members (e.g., an array with an array size of 3 having only two members). Any columns in the table marked as unique (via the table type manager) are checked for duplicate values. If a duplicate is found and updating is confirmed then the value is replaced with a blank.

Note that certain inconsistencies may prevent a complete check of a project. For example, if a column is missing from a data table then the table's data can't be loaded (an error dialog is displayed) and checked until the missing column is added at the end of the verification check. For this case the column should be allowed to be added during the first verification check, then a second verification performed so that the data within the affected table is checked. Inconsistencies ignored during the table data verification section may lead to subsequent inconsistency detection that otherwise wouldn't exist. An example would be ignoring a missing array definition when multiple array members are present – an issue is raised for each array member if the missing definition is ignored, whereas if updated the subsequent missing definition warnings won't occur.

While the check is being conducted a dialog appears (Figure 30) showing the verification progress. This dialog allows halting the verification by pressing the **Halt** button. No changes are made to the project database if the **Halt** button is pressed.

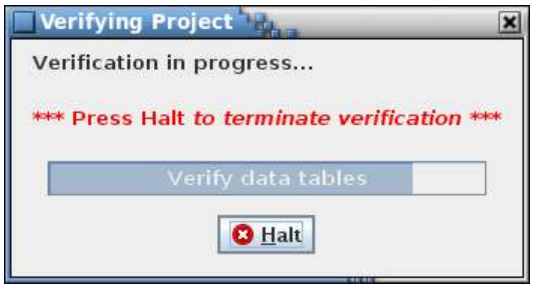


Figure 30. Verification and termination dialog

When the verification steps are complete, if any issues are detected then a dialog appears detailing the issues and the corrective action to be taken (Figure 31). Only users with administrative privileges (see paragraph 4.2.5) are allowed to implement the corrective actions (the **Okay** button is disabled for other users). After selecting the check box(es) in the **Corrective Action** column (or using the **Select all** check box to toggle selection of all of the issues), selecting **Okay** applies the corrective action(s) to the project database to address the issues flagged to be updated. Selecting **Print** allows outputting the list to a printer. Selecting **Cancel** exits the verification check without making any changes.

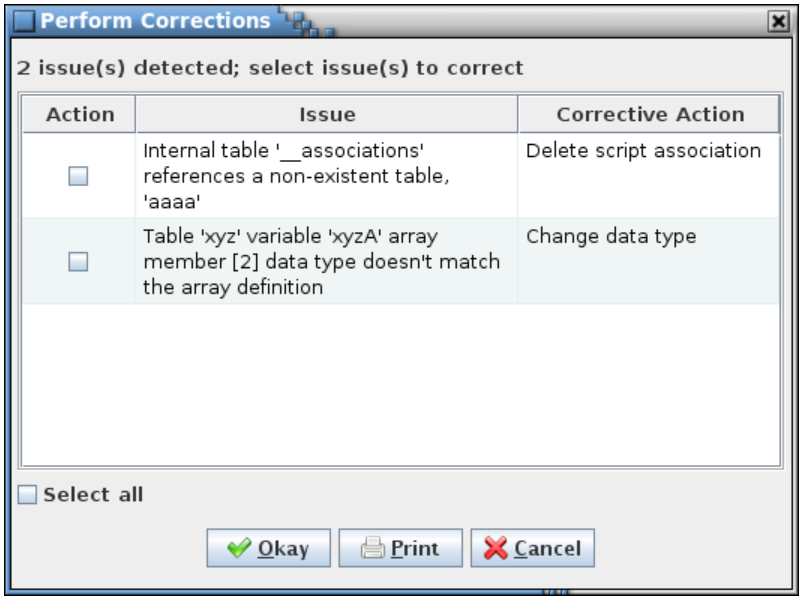


Figure 31. Example Perform Corrections dialog

4.9.2.11 Manage users

The Manage User Access Level dialog (Figure 32) provides a means to assign the level of access for each user of the current project. This command is enabled only for users with 'Admin' level access (see below). The creator of a project database is automatically granted administrator privileges. A user belonging to the project, but not explicitly assigned an access level, is granted 'Read Only' privileges.

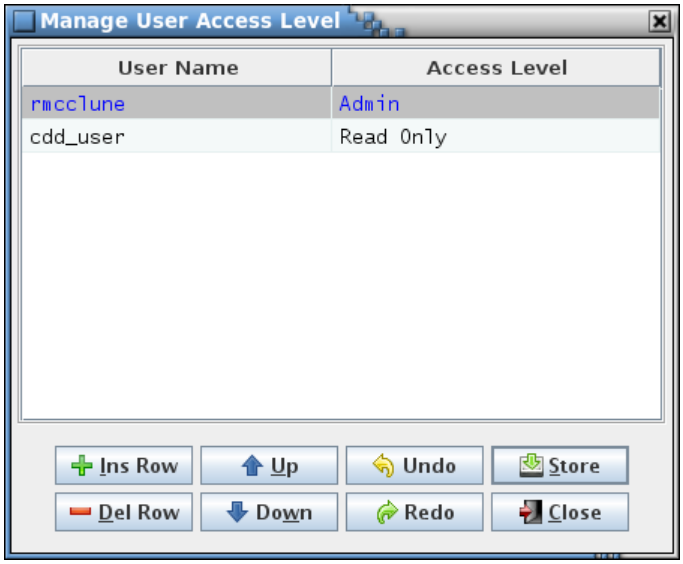


Figure 32. Manage User Access Level dialog

The editor column descriptions are as follows:

User Name This cell, when selected, displays a drop down menu containing all users registered in the PostgreSQL server. The current user is excluded from the list since a user can't change their own access level.

Acess Level This cell, when selected, displays a drop down menu containing the available access levels. These capabilities allowed by these levels are as follows:

Admin	In addition to the Read/Write access capabilities described below, the **Manage users** command is enabled so that the user may grant, alter, or remove a user's access to the project. Also, any issues detected during project verification (paragraph 4.9.2.10) may only be implemented by an administrator.
Read/Write	The user has full read/write access to the project database. All command menu and editor capabilities are enabled, except for the **Manage users** command. However, project verification updates may only be implemented by an administrator.
Read Only	The user has full read access to the project database, but all commands that can alter the database data are disabled. This doesn't prevent the user from altering the contents of any of the editors, but such changes are not persistent.

Each row in the table defines a user access level assignment. The rows can be sorted by selecting the column headers, as with other table editors in the application. Every assignment requires a value in the **User Name** and **Access Level** columns. The assignment for the current user is grayed out and can't be altered; only another user with administrative privileges can change or delete the current user.

The button commands are described below:

Ins Row	Inserts an empty row below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table.
Del Row	Deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this has no effect. The row containing the current user's access level (Admin by definition) cannot be deleted.
Up	Move the row(s) of the currently selected cell(s) up one row relative to the remaining rows. The order of the access level assignments in the editor has no effect on program operation. The capability to arrange the rows is solely for the user to group the users as desired.
Down	Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the access level assignments in the editor has no effect on program operation. The capability to arrange the rows is solely for the user to group the users as desired.
Undo	Undoes the last action performed (typing, paste, insert, delete, redo, etc.).
Redo	Reverses the last action undone (typing, paste, insert, delete, undo, etc.).
Store	Stores the changes made to user access levels into the database.
Close	Closes the user access level manager window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the dialog.

4.9.2.12 Recently opened project(s)

The remaining commands in the Project menu are the names of the most recently opened projects. Selecting one of these items opens the specified project. Note that the attempt to open the project is made in the currently attached PostgreSQL server.

4.9.3 Data

The **Data** menu has the commands for manipulating the data tables that contain a project's data.

4.9.3.1 New table(s)

The **New Table** command allows creation of a new data table. This command is enabled only for a user with read/write or administrator access. The **New Table** dialog (Figure 33) displays the defined table types and input fields for the table name and its description.

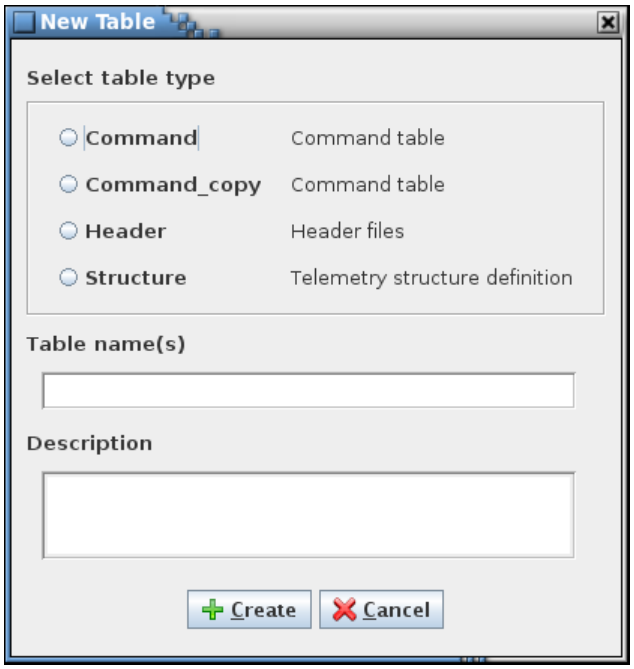


Figure 33. New Table dialog

A table type must be selected along with a valid table name. The description is optional and can be added or altered later using the table editor. Table names must be unique within a project. Though upper and lower case characters may be used, the name must still be unique if all of the characters are forced to lower case. The name must begin with a character or underscore (_) and can only contain characters, numerals, and underscores. Name length is constrained by PostgreSQL to a maximum of 63 characters. Also, the name may not match a primitive data type (e.g., double, or int8), a PostgreSQL reserved word, or begin with a pair of underscores (this is used to designate internal tables created by the CCDD application). A warning dialog appears if any constraint is violated.

Multiple tables of the same type may be created by entering more than one name in the table name field with each name separated by a comma. The new tables created in this manner share the description entered in the description field (if any). The descriptions can be added or altered later using the table editor.

Selecting **Create** causes the table(s) to be created and stored in the database. Each table created has the columns defined by the selected table type and initially has no rows. If the type chosen has default data fields, then the new table inherits these fields and their default values. The new table(s) can then be opened using the **Edit** command (see paragraph 4.9.3.2).

4.9.3.2 Edit table(s)

The **Edit table(s)** command displays the data table selection dialog (Figure 34).

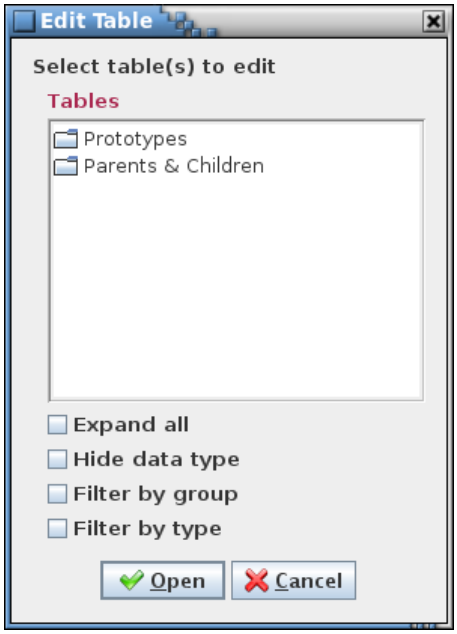


Figure 34. Edit Table dialog

The selection dialog has a table tree (see paragraph 4.5.3) from which one or more tables are selected for editing. Pressing the **Open** button opens the selected table(s) in a table editor (see Figure 35 for an example). Positioning the mouse pointer over a table name in the tree and double right-clicking can also be used to open the selected table. The **Cancel** button closes the table selection dialog without opening a table.

Menu bar
Table tab(s)
Table data
Description
Data field(s)
Buttons

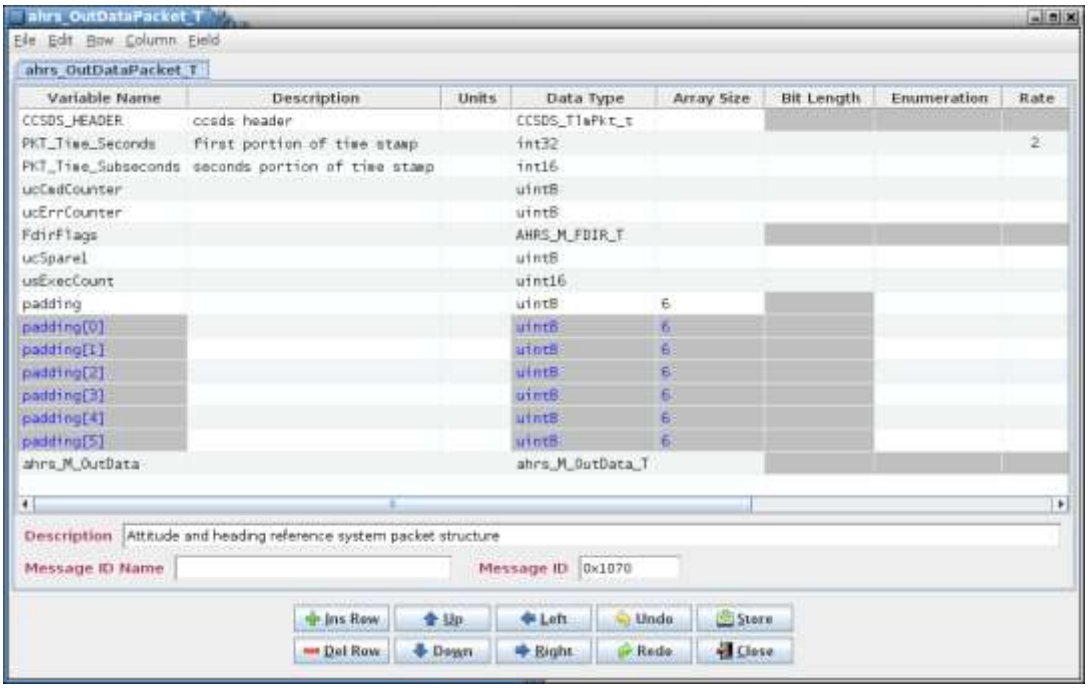


Figure 35. Example table editor

The table editor provides the means by which data is added to, altered, or removed from a data table. The editor is divided into six main sections.

Menu bar The first section is the menu bar, which contains the commands, described in the following paragraphs, for manipulating the table contents.

Table tab(s) The second section has one or more tabbed panes, each representing a data table's contents. The tab names indicate the table to which the tab applies. A prototype or root table shows only the prototype/root name. For a structure table's child table the tab displays a name in the format *root : structure.variable* where *root* is this table's root table name, *structure* is the name of the prototype structure represented by this table, and *variable* is the variable name that references this child table in the child's immediate parent structure. An asterisk beside the table name in the tab indicates that a change has been made to the table that hasn't been stored in the project database. Hovering the mouse pointer over the tab name produces a pop-up tool tip showing the table's type, root table, and complete structure and variable path.

Table data The columns displayed in the tabbed pane's table are determined by the table type of the table being edited. The table columns can be sorted and repositioned as described in paragraph 4.4; however, the sorted order does not dictate the actual table data row order. See paragraph 4.9.3.2.4 for the menu commands for repositioning the columns. If the column order change is stored in the database then it is restored when the table is reopened. Column ordering is preserved separately for each user.

Description This section contains the table description. The description is initially empty (unless set when the table was first created). The text entered here is used as a tool tip when the mouse pointer hovers over the table's name in the table tree. Letter, numeral, and punctuation characters may be entered. Additionally, HTML tags can be inserted to provide additional formatting to the tool tip text.

Data field(s) This section displays any data field(s) assigned to the table. See paragraph 4.6 for details concerning data field creation.

Buttons The remaining section has a series of buttons that perform some of the more commonly used commands. Certain buttons may be disabled depending on the table displayed in the editor. The buttons are as follows:

- **Ins Row** Inserts a new row in the table. See paragraph 4.9.3.2.3.1.
- **Del Row** Deletes the selected row(s) from the table. See paragraph 4.9.3.2.3.2.
- **Up** Moves the selected row(s) up one row. See paragraph 4.9.3.2.3.3
- **Down** Moves the selected row(s) down one row. See paragraph 4.9.3.2.3.4.
- **Left** Moves the selected column(s) left one column. See paragraph 4.9.3.2.4.1
- **Right** Moves the selected column(s) right one column. See paragraph 4.9.3.2.4.2.
- **Undo** Undoes the last action performed (typing, paste, insert, delete, redo, etc.).
- **Redo** Reverses the last action undone (typing, paste, insert, delete, redo, etc.).
- **Store** Stores the currently displayed table's contents (cell data, description, and data fields) in the database. See paragraph 4.9.3.2.1.3.
- **Close** Closes the currently displayed table's editor. See paragraph 4.9.3.2.1.8.

The following paragraphs provide details on the commands available in the table editor menu bar.

4.9.3.2.1 File menu

4.9.3.2.1.1 Edit table(s)

The **Edit table(s)** command displays the Edit Table dialog (Figure 34). The table(s) opened from this dialog appear in the current table editor under their own tabs.

4.9.3.2.1.2 Edit prototype

If the currently displayed table in the editor is a child table then issuing the **Edit prototype** command opens the child's prototype table in the current table editor under its own tab.

4.9.3.2.1.3 Store current

The **Store current** command stores the currently displayed table's contents, including the table's cell data, description, data fields, row order, and column order, into the database if changes have been made since the table was opened or since the last store operation. This command is enabled only for a user with read/write or administrator access. If no changes have been made then no action is taken; otherwise a confirmation dialog appears allowing the user to choose between continuing with the store operation and canceling it.

4.9.3.2.1.4 Store all

Selecting the **Store all** command is similar to the **Store current** command (paragraph 4.9.3.2.1.3) except that all tables in the table editor are stored to the database if changes have been made. This command is enabled only for a user with read/write or administrator access. A confirmation dialog appears allowing the user to choose between continuing with the store operation and canceling it.

4.9.3.2.1.5 Import data

The **Import data** command provides a means of inserting data and/or data fields from a CSV, EDS XML, JSON, or XTCE XML formatted file into the table currently displayed in the table editor. The import file may contain information for multiple tables, but only the data (and fields for CSV and JSON imports) for the first table defined in the file are imported. A dialog appears (Figure 36) allowing the user to choose the import file based on file format. The **Export table** and **Export table(s)** commands produce files compatible with the import command; see paragraphs 4.9.3.2.1.6 and 4.9.3.8; Appendix C contains details on the file formats.

The **Overwrite existing cells** check box, if selected, causes the imported data to overwrite data in the existing cells, beginning at the currently selected row. If unchecked, rows are inserted into the table at the currently selected row (or at the end of the table if no cell is selected) to contain the imported data.

The **Use existing field if duplicate** check box, if selected (the default), causes any imported data fields to be ignored if the table already has a field of the same name. If unchecked, then the existing data field is replaced by the imported one.

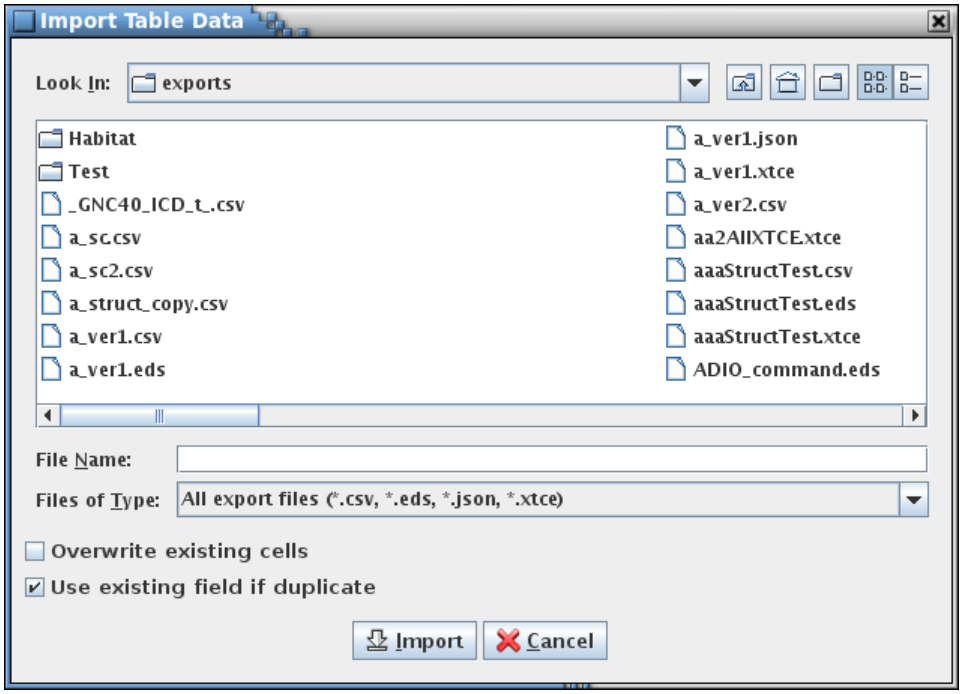


Figure 36. Import data and fields into an existing table dialog

For CSV and JSON imports the data type and macro definitions in the import file, if present, are ignored. The table type definitions are evaluated in the event the table from which the data is imported is otherwise undefined. If a column name is missing in the table type definition a dialog appears alerting the user to the error. The user can elect to ignore this table type definition and continue with the import, ignore any remaining table type errors in this import file and continue, or cancel the import. Following the import any table type additions are removed.

For CSV and JSON imports the column names in the import file associated with the row data are compared to those in the table's table type definition columns in order to insert the cell values into the proper column. If the column name associated with a cell value isn't recognized as one belonging to the table's table type definition then that cell value is ignored and a dialog appears alerting the user to the discrepancy. The user can elect to ignore the unrecognized column name and continue with the import, ignore any remaining unrecognized column names in this import file and continue, or cancel the import. For EDS AND XTCE imports the data is placed in the structure or command table based on the table's table type definition column input types (for example, a variable name is placed in the column with the **Variable name** input type, regardless of its position in the table).

4.9.3.2.1.6 Export table

The **Export table** command provides a means of outputting the current table's definition to a file in CSV, EDS, JSON, or XTCE format. This is equivalent to the **Export table(s)** command in the main window's **Data** menu - see paragraph 4.9.3.8 for details. The dialog that appears when is similar to that for the **Export table(s)** command, except there is no table tree.

4.9.3.2.1.7 Print current

The **Print current** command outputs the currently displayed table to a printer or file selected by the user from the printer dialog that appears. The table's data fields are also output on a separate page.

4.9.3.2.1.8 Search

Selecting the **Search** command causes a search dialog to be displayed (see Figure 37). This dialog allows searching for matching text in the currently active table editor. The search dialog remains on top of the table editor dialog. If the editor dialog has multiple tabs, then when another tab is selected the search criteria are applied to the new active table editor. If a window other than that for the table editor dialog or search dialog is selected the search dialog is hidden; the dialog becomes visible again when the table editor dialog is selected. Pressing the Ctrl-F key sequence while the table editor dialog window has the focus also displays the search dialog.

Case sensitivity for the search is governed by the **Ignore text case** check box.

The **Allow regular expression** check box, when checked, allows the use of a regular expression to define the search pattern in the search text field. A regular expression can be formulated to find multiple matching conditions (for example, the search for **a.c** would match any string that has a single character between the characters 'a' and 'c'). Information regarding the use of regular expressions is beyond the scope of this document; however, resources and tutorials can be found online. When unchecked, the search text is matched as typed in the search text field.

Enter the search text in the input field and select the **Search** button. Any matching text in the table is highlighted and the total number of matches, including multiple matches in a single cell, is displayed beside the search text field's label. If the search text field is empty and the **Search** button is pressed, or if the search dialog is closed, then any search match highlighting is removed.

The search text field uses auto-completion to fill in the search string. The previous search strings (those for the event log, table, and script) are remembered, including those from previous sessions. The number of remembered search strings can be changed via the Preferences dalog, and defaults to 30. Case sensitivity for auto-completion is based on the **Ignore text case** check box selection state.

If the table contains one or more instances of the search text then the search dialog's **Previous** and **Next** buttons are enabled. These buttons cause the table cell selection to move from the currently selected cell to the previous or next cell containing a search match, wrapping to the end or beginning of the table if needed. The table scrolls, and for a structure table array members are expanded, if necessary for the selected cell's row to be visible.

The input text can be changed and the **Search** button pressed again to initiate another search of the table. To exit the search dialog select the **Close** button.

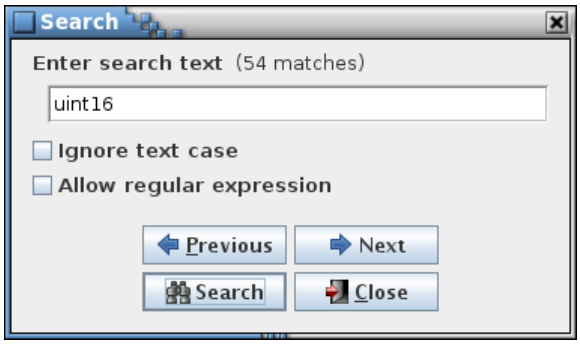


Figure 37. Data table search dialog

4.9.3.2.1.9 Close current

The **Close current** command closes the currently displayed table's editor tab. If this is the last table in the editor then the editor window is also closed. If any changes to the table's data, description, data fields, row order, or column order have been made, but not stored in the database, then a confirmation dialog appears allowing the user to choose between continuing with the close operation, discarding the changes, or canceling it, keeping the table editor open.

4.9.3.2.1.10 Close all

The **Close all** command performs a similar operation to the **Close current** command (paragraph 4.9.3.2.1.8) except all tables in the editor are closed as well as the editor. For unstored changes in any of the tables, only a single confirmation dialog appears; if confirmed, the changes in all tables in the editor are discarded.

4.9.3.2.2 Edit menu

4.9.3.2.2.1 Copy

The **Copy** command places the contents of the highlighted cell(s) into the operating system's clipboard. This information can then be pasted into another table cell or input field within the application, or into applications other than CCDD. The Ctrl-C keys perform the same operation.

4.9.3.2.2.2 Paste

The **Paste** command places the contents of the operating system's clipboard into one or more table cells, if a cell is selected, or into an input field if a field is selected. The paste location within the table is determined by the leftmost and uppermost highlighted cell. If the table contains collapsed arrays then the arrays are expanded prior to pasting the data. The rows and columns of the copied cells are then placed into the editor beginning at this location and extending down and to the right, overwriting the existing data in the cells. If insufficient columns exist for the pasted data then the excess column data is ignored. Extra rows are inserted at the bottom of the table to provide room for data that would be placed below the editor's last row. See paragraph 4.9.3.2.2.3 on inserting copied data without overwriting the existing cell contents. If data from multiple cells is pasted into an input field then content from each cell is concatenated, separated by a space, and the result is pasted into the field. The Ctrl-V keys perform the same operation.

4.9.3.2.2.3 Insert

The **Insert** command behaves similarly to the **Paste** command (paragraph 4.9.3.2.2.2) except that no editor data is overwritten. Instead, rows are inserted, beginning at the row below the upper- and leftmost highlighted cell, to accommodate the pasted values. The Ctrl-I keys perform the same operation.

4.9.3.2.2.4 Undo

The **Undo** command performs the same action as the **Undo** button. The command undoes the last action performed (typing, paste, insert, delete, redo, etc.) in the currently displayed table editor. The Ctrl-Z keys perform the same operation.

4.9.3.2.2.5 Redo

The **Redo** command performs the same action as the **Redo** button. The command reverses the last action undone (typing, paste, insert, delete, redo, etc.) in the currently displayed table editor. The Ctrl-Y keys perform the same operation.

4.9.3.2.2.6 Insert macro

The **Insert macro** command provides a convenient means to insert a macro into a table cell. Editing must first be initiated in the cell. Position the text cursor within the cell at the point where the macro is to be inserted, or select any existing text the macro is to replace, then select the **Insert Macro** command. A pop-up list appears displaying all macros with values that are consistent with the input type of the cell being edited (see Figure 38; see paragraph 4.7 for information on input types). Use the mouse or keyboard to highlight the macro to insert. If the mouse pointer is hovered over a macro name in the pop up a tool tip pop up appears displaying the macro's value. Once the desired macro is highlighted either press the left mouse button or the Enter key. The macro name is inserted into the table cell, replacing any selected text, bounded by the macro delimiter characters (##), and highlighted to aid in distinguishing it from the non-macro text (see Figure 38). Press the Escape key to remove the macro pop up dialog without inserting a macro.

The Ctrl-M keys can be used in place of the **Insert macro** command. While editing a cell, position the text cursor or highlight one or more characters to be replaced, then press Ctrl-M. The pop-up list appears as described above.

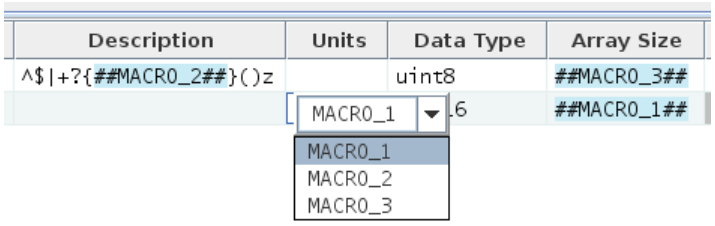


Figure 38. Example of macro name display and pop-up dialog in a data table

4.9.3.2.2.7 Show macros

The **Show macros** command, when checked, temporarily replaces every macro with its corresponding value in each table of the editor dialog. No editing of the table may take place while the command is selected (all tables cells are grayed out and most of the editor menu commands and buttons are disabled). Deselecting the **Show macros** command restores the macros and enables normal editing and commands.

Pressing Ctrl-Shift-M produces a similar effect, causing the currently selected table to display all cells containing a macro to replace the macro with its value; releasing the Ctrl-Shift-M keys restores the macro names in the cells.

4.9.3.2.2.8 Clear selected

The **Clear selected** sub-menu contains commands for replacing the contents of the selected cell(s) in the currently displayed table.

4.9.3.2.2.8.1 With blanks

The **With blanks** command replaces the values in the selected cell(s) with a blank. Pressing the Delete key performs the same action.

4.9.3.2.2.8.2 With prototype

The **With prototype** command is only available if the table editor is displaying a child table. This command replaces the value in selected cell(s) with the contents of the corresponding cell in the table's prototype table. Pressing the Shift-Delete keys performs the same action. A special indicator flag (Ø) is prepended to the cell value and highlighted (see Figure 39 for an example). If the table changes are stored or the cell is subsequently edited this indicator is removed.

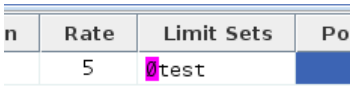


Figure 39. Special indicator flag example

4.9.3.2.3 Row menu

4.9.3.2.3.1 Insert row

The **Insert row** command performs the same action as the **Ins Row** button. The command causes an empty row to be inserted below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table. The Insert key performs the same operation.

A row may not be inserted within an array. If an array definition or member cell is selected then the new row is inserted below the last array member.

4.9.3.2.3.2 Delete row(s)

The **Delete row(s)** command performs the same action as the **Del Row** button. This command deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this command has no effect. The Delete key performs the same operation.

A row may not be removed from within an array. If an array definition or member cell is selected then the entire array – definition and members – is deleted.

4.9.3.2.3.3 Move up

The **Move up** command performs the same action as the **Up** button. This command causes the row(s) of the currently selected cell(s) to move up one row relative to the remaining rows. Only prototype tables may have their rows reordered; reordering the prototype's rows affects the row ordering for all tables based on the prototype. Reordering the rows is recognized as a table change and is preserved in the database via use of the **Store** command (menu or button).

Arrays cannot be split by use of the **Move up** command. If an array definition or member cell is selected then the array moves up as a unit.

4.9.3.2.3.4 Move down

The **Move down** command performs the same action as the **Down** button. This causes the row(s) of the currently selected cell(s) to move down one row relative to the remaining rows. Only prototype tables may have their rows reordered; reordering the prototype's rows affects the row ordering for all tables based on the prototype. Reordering the rows is recognized as a table change and is preserved in the database via use of the **Store** command (menu or button).

Arrays cannot be split by use of the **Move down** command. If an array definition or member cell is selected then the array moves down as a unit.

4.9.3.2.3.5 Expand arrays

The **Expand arrays** command is only available for those tables containing a column with an input type of "Array index" (e.g., the **Array Size** column in the default **Structure** table type). This command toggles display of array members in the table. When enabled, each array member is displayed in its own row in the table beneath the array's definition row. When disabled, the array members are hidden, though the array's definition row continues to be displayed. Array member visibility can also be toggled by positioning the mouse pointer over any cell in the array size column (except the column header) and double right-clicking.

4.9.3.2.3.6 Array overwrite

The **Array overwrite** command is a submenu with three mutually exclusive selections: **Overwrite all**, **Overwrite empty**, and **Overwrite none**. The selection governs pasting of data into array member cells already containing values. **Overwrite all**, the default, overwrites any existing values with the pasted values. **Overwrite empty** only pastes values into cells that are currently empty; paste values are discarded if the target cell is occupied. **Overwrite none** prevents pasting values into array member cells.

4.9.3.2.4 Column menu

4.9.3.2.4.1 Move left

Issuing the **Move left** command moves the column(s) of the selected cell(s) to the left one column relative to the remaining columns. Reordering the columns is recognized as a table change and is preserved in the database separately for each user via use of the **Store** command (menu or button).

4.9.3.2.4.2 Move right

Issuing the **Move right** command moves the column(s) of the selected cell(s) to the right one column relative to the remaining columns. Reordering the columns is recognized as a table change and is preserved in the database separately for each user via use of the **Store** command (menu or button).

4.9.3.2.4.3 Reset order

The **Reset order** command restores the column order for the currently displayed table to the default order. The default order is established by the order of the column definitions in the table type editor for the table's type.

4.9.3.2.5 Field menu

4.9.3.2.5.1 Manage fields

The **Manage fields** command allows the user to create, alter, and delete data fields for the table represented by the active table editor tab. See paragraph 4.6 for information regarding data fields and use of the data field editor.

The fields manipulated by the field editor are displayed below the table editor table and description when the **Update** button is pressed. After the field editor is closed values can be entered into the data fields. The table editor's **Store** button or command must be used to store the changes in the project database and apply them to the table.

4.9.3.2.5.2 Clear values

The **Clear values** command clears the contents of all of the currently displayed table's data fields. A confirmation dialog is first displayed. Selecting **Okay** causes all of the data field values to be blanked. Selecting **Cancel** exits the dialog without affecting the data field values.

4.9.3.3 Recent

This sub-menu contains the names (with paths) of the most recently opened tables. Selecting one of these items opens the specified table. Note that the attempt to open the table is made in the currently open project.

4.9.3.4 Rename table

The **Rename table** command allows a prototype data table to be renamed. This command is enabled only for a user with read/write or administrator access. Child tables cannot be renamed using this dialog. Child tables are instances of a prototype table assigned as a variable, so a child table's name is a combination of its prototype table name and the variable name in its parent table's prototype.

Therefore, child table names can only be altered by changing the name of the child table's prototype table, or changing the name of the variable representing the child table in its parent table's prototype table.

The **Rename Table** dialog (Figure 40) appears, displaying a table tree showing the prototype tables and input fields for providing the selected table's new name and description. See paragraph 4.5.3 for details on the table tree. A table is first selected from the tree; the table name and description (if any) appear in the input fields. After altering the name and description fields as desired the **Rename** button is selected to change the table's name and description. See paragraph 4.9.3.1 for details on table name constraints. The new name and description are immediately reflected in all parent and child tables, including those appearing in open table editors. The description is optional and can be added or altered later using the table editor. Select the **Cancel** button to exit the dialog without making a change to a table name.

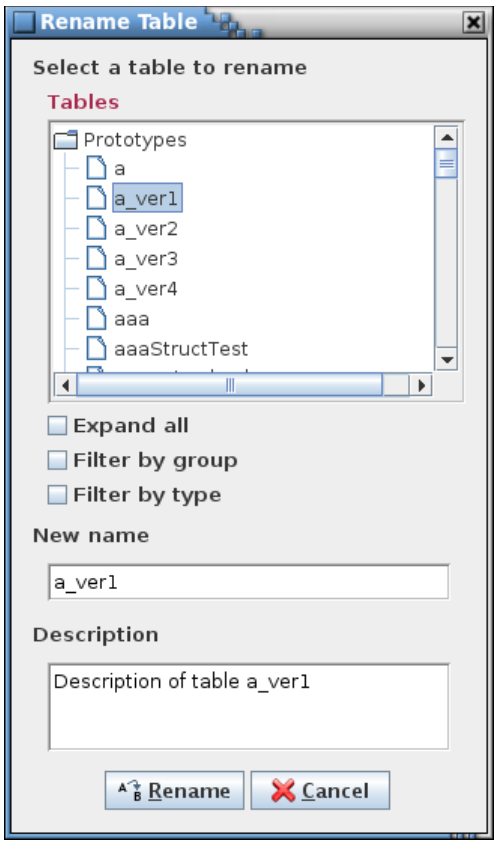


Figure 40. Rename Table dialog

4.9.3.5 Copy table

The **Copy table** command allows a prototype data table to be copied. This command is enabled only for a user with read/write or administrator access. The **Copy Table** dialog (Figure 41) appears, displaying a table tree showing the prototype tables and input fields for providing the name and description of the selected table's copy. See paragraph 4.5.3 for details on the table tree. A table is first selected from the tree; the table name appears in the input field with "_copy" appended and its description, if any, appears in the description field. After altering the name and description fields as desired the **Copy** button is selected to create the table's copy. See paragraph 4.9.3.1 for details on table name constraints. The description is optional and can be added or altered later using the table editor. Select the **Cancel** button to exit the dialog without copying a table.

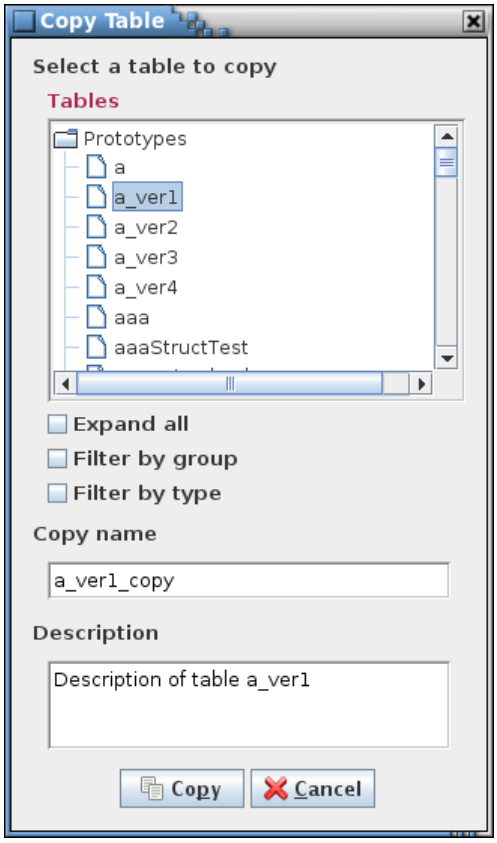


Figure 41. Copy Table dialog

4.9.3.6 Delete table(s)

The **Delete table** command allows one or more prototype tables to be deleted. This command is enabled only for a user with read/write or administrator access. The **Delete Table** dialog (Figure 42) appears, displaying a table tree showing the prototype tables. See paragraph 4.5.3 for details on the table tree. After one or more tables is selected from the tree the **Delete** button is selected to delete the table(s). A confirmation dialog appears and if **Okay** is selected all instances of the deleted table, both parent and child tables, are deleted from the project database, including those appearing in open table editors. Select the **Cancel** button to exit the dialog without deleting a table.

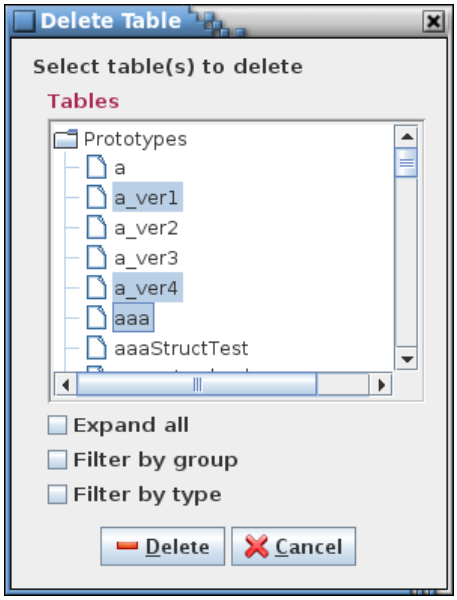


Figure 42. Delete Table dialog

4.9.3.7 Import table(s)

The **Import table(s)** command allows importing information from CSV, EDS, JSON, and XTCE formatted files in order to create tables in the project database. This command is enabled only for a user with read/write or administrator access. CSV and JSON import files can contain information for tables of any type (structure, command, and other user-defined types), table type definition(s), data type definition(s), macro definition(s), reserved message IDs, and variable paths. EDS and XTCE XML import files are restricted in the information contained, so only basic information for structure and command tables can be imported from files in these formats. When the command is selected a dialog appears allowing the user to choose the location of the import file(s) (see Figure 43).

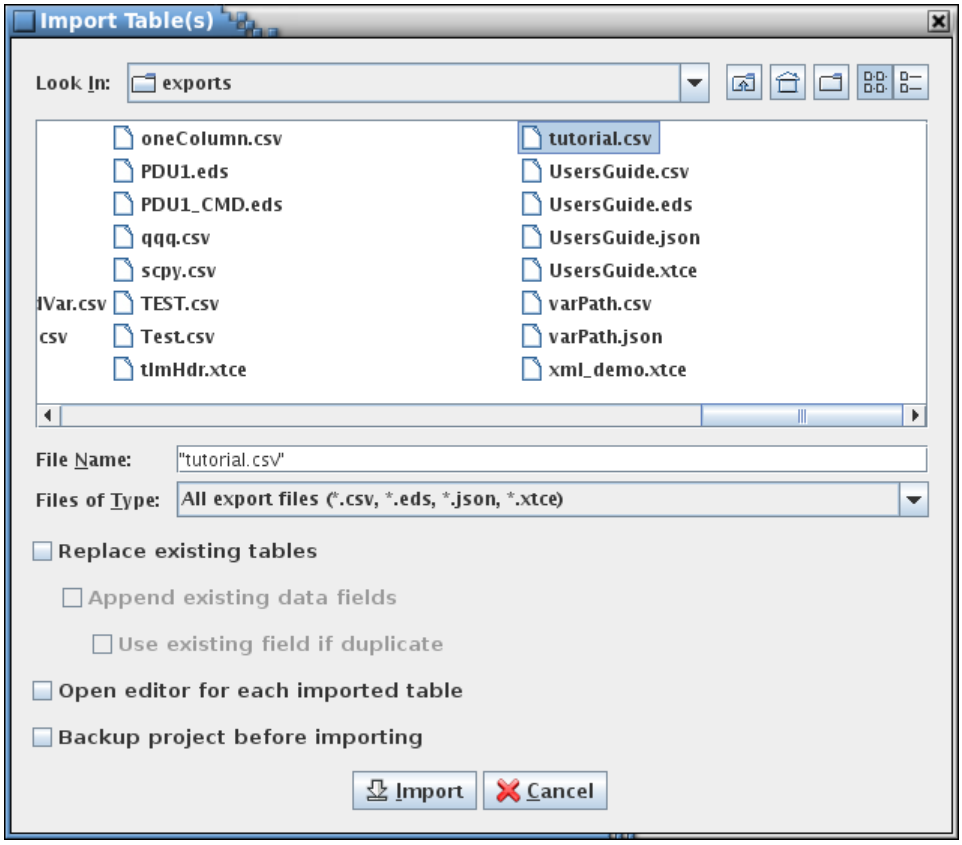


Figure 43. Import table(s) dialog

The **Replace existing tables** check box allows the user to choose whether to replace any existing tables with the same name with the import file data. If selected, the **Append existing data fields** check box is enabled, which indicates if replaced tables that have data fields have the existing fields appended to those imported. If this check box is selected then the **Use existing field if duplicate** check box is enabled, which determines whether to use the existing data field or the imported one in the event that the data fields have the same name.

The **Open editor for each imported table** check box, if selected, causes a table editor dialog to be opened, and a table editor displayed for each table imported from the file. If a large number of tables is imported, then multiple table editor dialogs may be created, based on the number of rows of tabs taken up by the table names in the dialog. The maximum number of tab rows is user-adjustable via the **Preferences** dialog (see paragraph 4.9.1.7.4); the default is 15.

The remaining check box, **Backup project before importing**, determines if a backup of the project database is created prior to continuing with the import operation.

More information regarding the permissible formats for the import file are described in Appendix C.

When a structure table is imported, if it has a variable with a data type that is a reference to another structure and the prototype for this child structure doesn't already exist in the project and isn't defined in the import file(s), then the prototype table is created. This auto-generated prototype is a copy of the child structure; i.e., it's created using the same structure table type definition as the child structure and contains all of the information in the child.

4.9.3.7.1 Importing table types

When importing from EDS or XTCE XML formatted files a single structure and/or a single command table type is created on which all of the structure and command tables are based. If more than one command

table definition is present in the import file then the one with the largest number of arguments determines the command table type definition (these argument entries are empty for command tables requiring fewer arguments). The table type definition names are 'Structure' and 'Command' with with 'EDS_' or 'XTCE_' prepended and are guaranteed to be unique (a numeral is appended to the table type definition name as needed to differentiate the new type from an existing one).

CVS and JSON formatted files must either use a table type definition already defined in the target project database, or contain a definition for each type of table imported. If a table type present in the import file has the same name as one already present in the project then the column definitions must match (other than for the column descriptions), otherwise an error dialog appears; if the error is ignored then this table type definition is skipped. Note that any tables in the import file using this type won't successfully be imported. Three options are available to overcome the table type collision: (1) change the existing type's name, (2) change the type's name in the import file (including any references in any table definitions using it), or (3) alter the existing table type to match the imported type (adding, removing, or altering the column definitions as needed). Once the changes are made the import can be performed again.

When importing a table type column definition the inputs are expected in the same order as they appear in the table type editor (see Figure 50). If the number of inputs is fewer or more than those expected, or the column name is blank, then an error dialog appears; if the error is ignored then this table type definition is skipped. If the input type is blank then it defaults to 'Text', but if the input type is present and isn't one recognized by the application (see paragraph 4.7) a warning error dialog appears; if the error is ignored then the input type 'Text' is used. The inputs representing the check box columns evaluate to 'true' if the input is 'true' (case insensitive); otherwise 'false' is used, regardless of the actual value supplied.

4.9.3.7.2 Importing data fields

When importing from EDS and XTCE XML formatted files certain data fields are created if the information is present in the import file. These fields are the application (or message) ID for root structure tables and command tables, and the system path for each table (root or instance).

When importing a data field definition from a CSV or JSON formatted file the inputs are expected in the same order as they appear in the data field editor (see Figure 7). This includes an input for the "Applicability Type", even if the data field isn't associated with a structure table type definition. An additional input containing the field's value is expected as the last input. If the number of inputs is fewer or more than those expected, or the field name is blank, then an error dialog appears; if the error is ignored then this data field is skipped. If the input type is blank then it defaults to 'Text', but if the input type is present and isn't one of the recognized ones a warning dialog appears; if the error is ignored then the input type 'Text' is used. The input representing the check box column evaluates to 'true' if the input is 'true' (case insensitive); otherwise 'false' is used, regardless of the actual value supplied. If the applicability type is blank then it defaults to 'All tables', but if the applicability type is present and isn't one of the recognized ones a warning dialog appears; if the error is ignored then the applicability type 'All tables' is used.

4.9.3.8 Export table(s)

The **Export table(s)** commands allow outputting a project's data table information in one of four formats: CSV, JSON, EDS XML, and XTCE XML. The CSV and JSON formats include information for the table type definition(s), data type definition(s), macro definition(s), reserved message IDs, variable paths, and table definition(s). This allows these formats to exchange detailed project information with another CCDD project. The XML formats only export a subset of the information that appears in the project's structure and command data tables.

Each export dialog, shown in the following sections, differs in options and inputs. The common features are as follows. At the top of the dialog is a table tree for selecting the tables to export; below the tree are the various tree filters – see paragraph 4.5.3 for more details. The dialog includes an input field for specifying the export path or file. The CSV and JSON formats allow for storing the output either combined into a single file or with each table in its own file based on the selection of the radio buttons (**Store in Multiple files** or **Single file**). The EDS and XTCE output is to a single file only. The output file name or path must be entered in the export file/path field, or selected from the file chooser dialog that appears when the **Select…** button is pressed. For multiple tables with output to individual files the file names are automatically assigned based on the table names. If an output file already exists then the table isn't exported unless the **Overwrite existing file(s)** check box is selected or the user confirms overwriting each file (via a dialog that appears during the export operation).

The dialogs for the EDS and XTCE exports contain a pair of radio buttons, **Endianess Big** and **Little**, that are used to determine the flags set in the export file that indicate if the data is in big or little endian format. If little endianness is selected then the check box **Headers are big endian** is enabled. If selected, the telemetry and command header table variables are flagged as big endian (as an example, the CFS CCSDS headers are always big endian). This check box is selected by default.

The following inputs allow for modifying what other information is included in the export file and only apply to CSV and JSON exports. When the **Substitute macro values for macro names** check box is selected the table cells containing macro references have the references replaced with the corresponding macro value. The macro definitions are not stored in the export file(s) in this case. The **Include reserved message IDs** check box, when checked, causes the message IDs in the reserved message ID editor (see paragraph 4.9.3.14.2) to be exported. If the **Include project data fields** check box is selected then all of the project-level data field definitions are included in the output. If the **Include variable paths** check box is selected then the variable path, in both application and user-defined formats, for each variable is included in the output. The remaining inputs become active when this check box is selected. These inputs behave identically to those described in paragraph 4.9.3.18.

The column data in an exported table file may be imported into an existing table using the **Import data** command (paragraph 4.9.3.2.1.5). The **Import table(s)** command (paragraph 4.9.3.7) imports the entire contents of the export file into the current project, creating tables, table type definitions, and data field definitions (and data type definitions, macros, reserved message IDs, and variable paths in the case of CSV and JSON formatted files) as described in the file. Details on the allowable formats of the export file are described in Appendix C.

4.9.3.8.1 CSV

The **Export table(s) - CSV** command allows exporting one or more selected tables in CSV format. See Appendix C.1 for details on the file format.

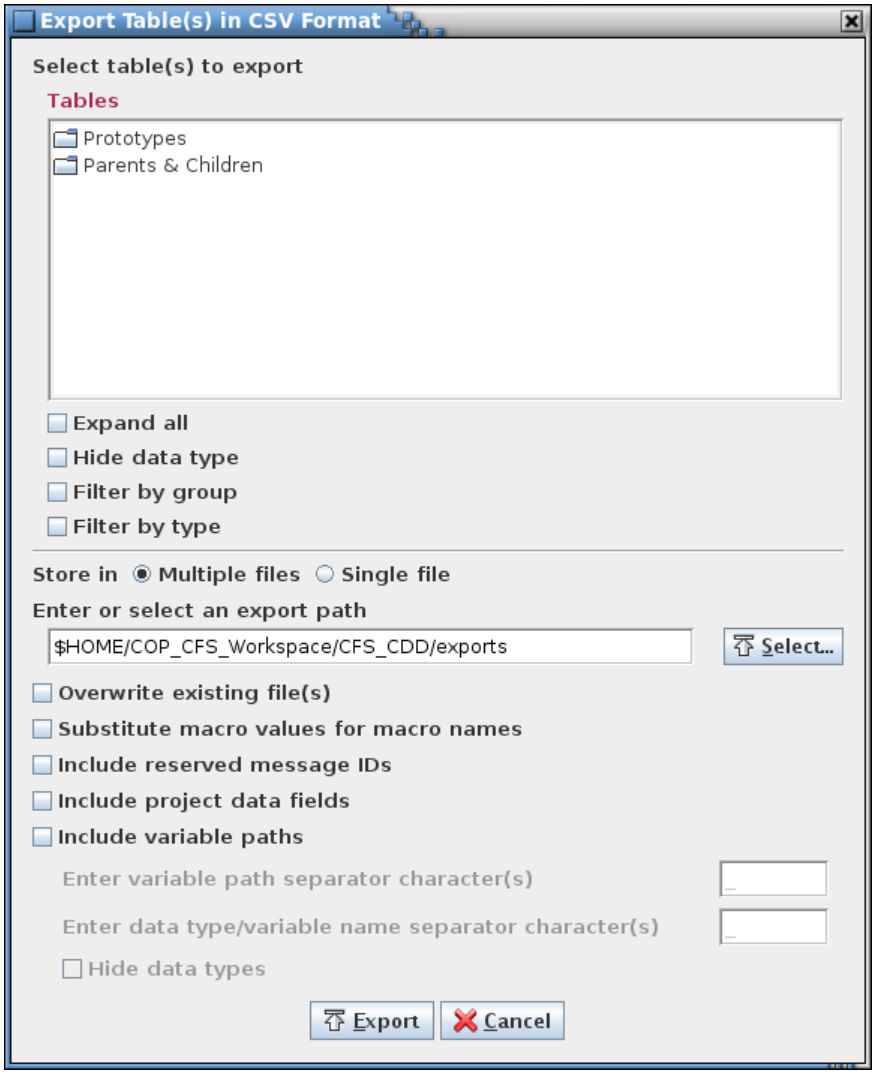


Figure 44. CSV export dialog

4.9.3.8.2 EDS

The **Export table(s) - EDS** command allows outputting one or more selected tables in EDS XML format. Only certain data for structure and command tables is exported; other table types are ignored. Any macros are expanded in the exported data. See Appendix C.2 for details on the file format.

Four special project-level data fields are utilized by the EDS format (see paragraph 4.9.3.15 for information on assigning project-level fields). These fields are described in Table 7. The names of the fields can be whatever the user desires; the input types determine how the fields are used during the export process. The telemetry and command header table names indicate which tables represent the telemetry and command header structures, respectively. All root structure tables reference a single telemetry header in the export file, and all command tables reference a single command header in the export file. If a field is missing or its contents is empty the export is still performed, but certain data may be missing. For example, without the header fields or the application ID and function code fields the export operation can't determine where to place the ID and function code values in the export file, so these values are ignored. The contents of the four data fields are stored as device metadata in the export file. Application ID and command function code use default values if not specified in the data fields.

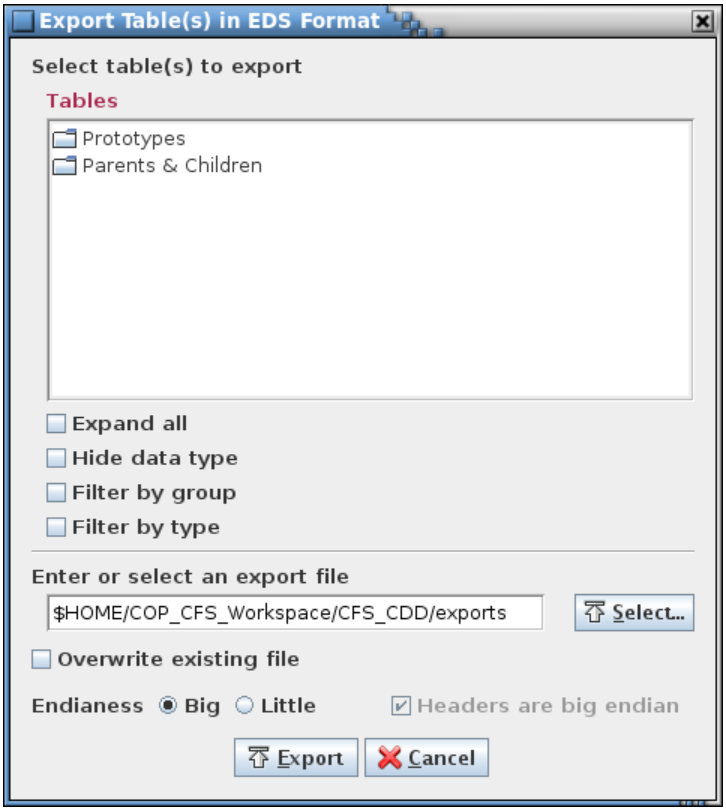


Figure 45. EDS export dialog

Input Type	Description
XML: Application ID	Name of the variable containing the application ID in the structure tables representing the telemetry and command headers
XML: Command Header	Name of the structure table representing the command header
XML: Function Code	Name of the variable containing the command function code in the structure table representing the command header
XML: Telemetry Header	Name of the structure table representing the telemetry header

Table 7. XML special data fields

4.9.3.8.3 JSON

The **Export table(s) - JSON** command allows outputting one or more selected tables in JSON format. See Appendix C.3 for details on the file format.

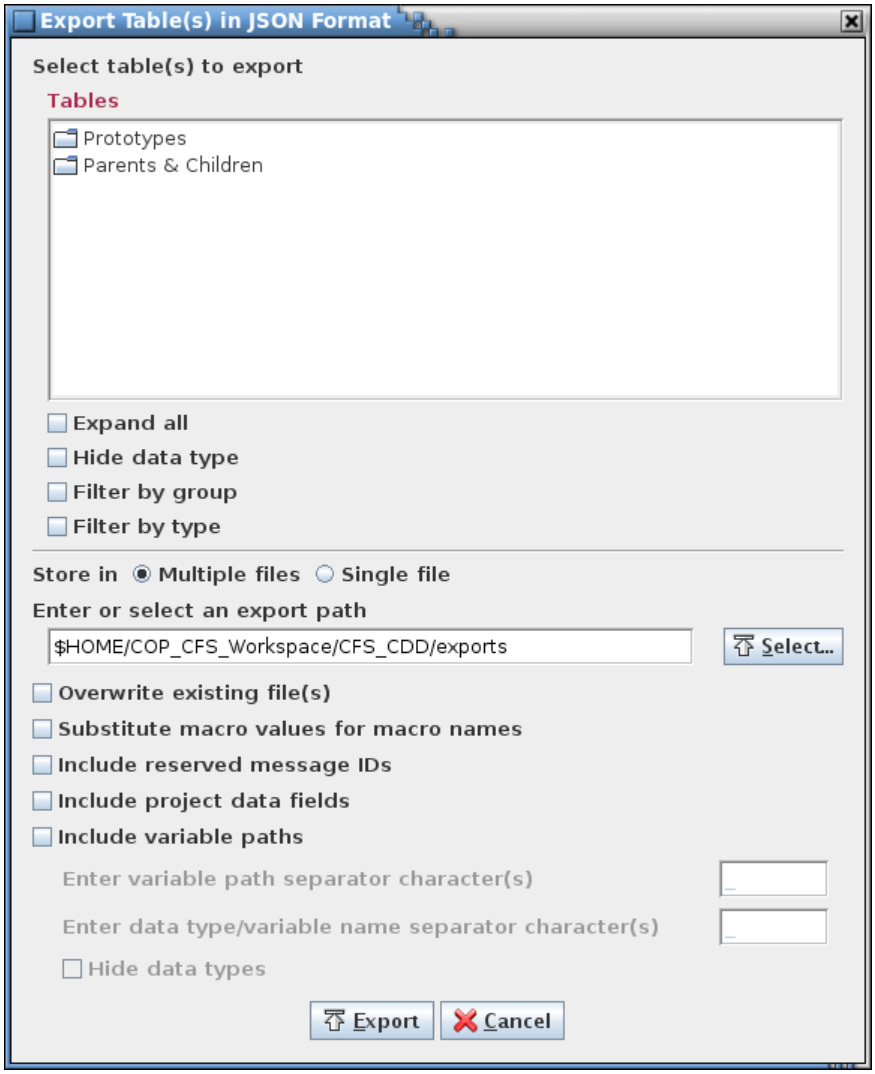


Figure 46. JSON export dialog

4.9.3.8.4 XTCE

The **Export table(s) - XTCE** command allows outputting one or more selected tables in XTCE XML format. Only certain data for structure and command tables is exported; other table types are ignored. Any macros are expanded in the exported data.

The **Use external methods** check box, when checked, indicates that the script, named in the **Enter script file name** input field, is used to perform some or all of the XTCE conversion. The process is identical to performing the export operation via script association (see paragraph 4.10.7 for details on modifying the XTCE conversion via a script). However, not all of the script data access methods are available when using the external methods feature from the export dialog – any method referencing the table names, rows, table data, etc. is not accessible.

The XTCE dialog has additional inputs for defining certain XTCE attributes and classification levels that are used when constructing the output file. Default values are provided, but can be changed as desired. The attributes are used to construct the XML **Header** elements. See Appendix C.4 for details on the file format.

Four special project-level data fields are utilized by the XTCE format (see paragraph 4.9.3.15 for information on assigning project-level fields). These fields are described in Table 7. The names of the

fields can be whatever the user desires; the input types determine how the fields are used during the export process. The telemetry and command header table names indicate which tables represent the telemetry and command header structures, respectively. All root structure tables reference a single telemetry header in the export file, and all command tables reference a single command header in the export file. The command tables all reference a single command header in the export file. If a field is missing or its contents is empty the export is still performed, but certain data may be missing. For example, without the header fields or the application ID and function code fields the export operation can't determine where to place the ID and function code values in the export file, so these values are ignored. The contents of the four data fields are stored as ancillary data in the root system in the export file. Application ID and command function code use default values if not specified in the data fields.

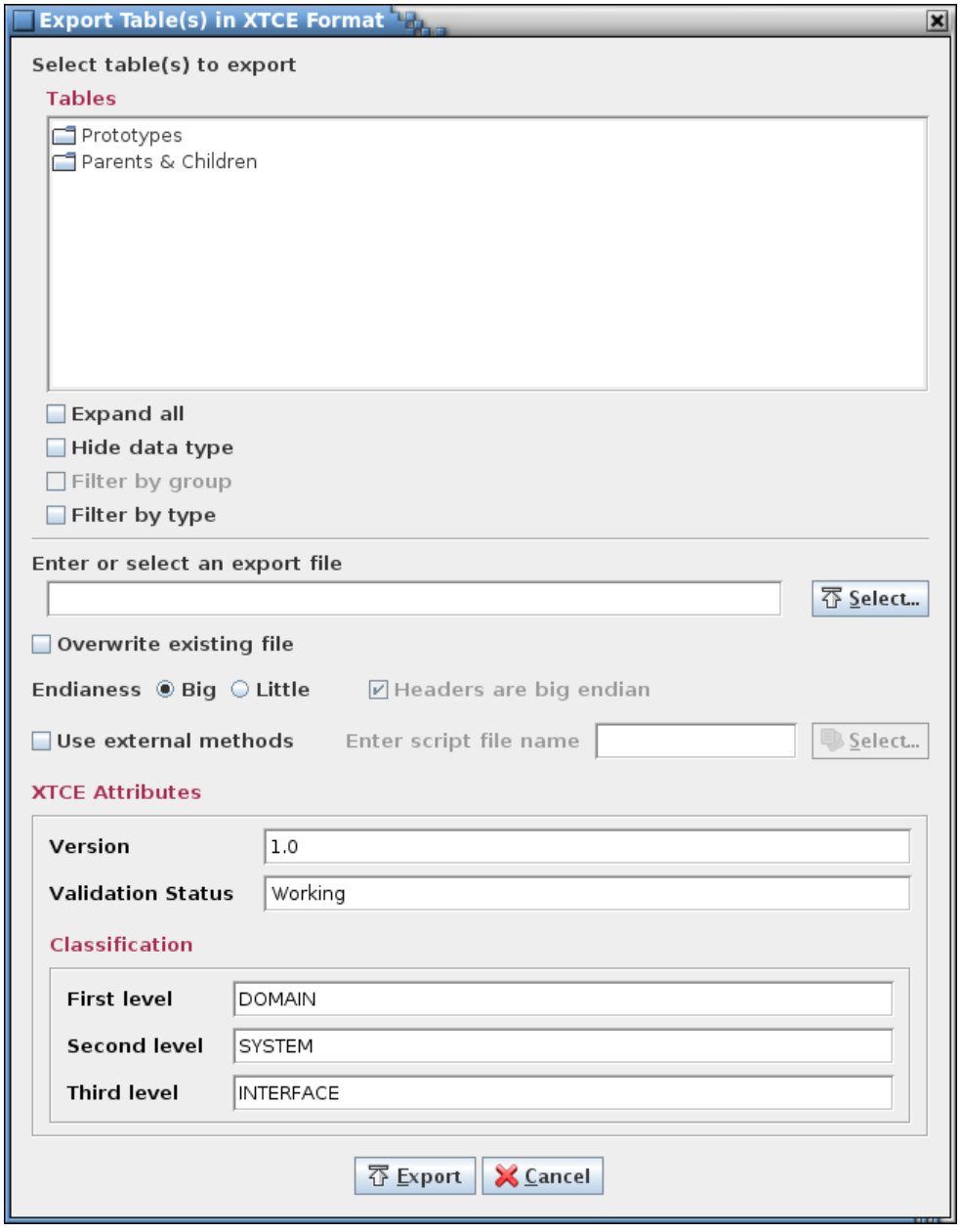


Figure 47. XTCE export dialog

4.9.3.9 Manage groups

The **Manage groups** command allows data tables to be assigned to user-defined groups. These groups can be used to filter the tables in the table trees used in other dialogs, making it easier to locate tables that are related (e.g., by a vehicle subsystem or CFS application). Groups can be added, altered, or deleted. When the command is selected a dialog similar to that in Figure 48 appears.

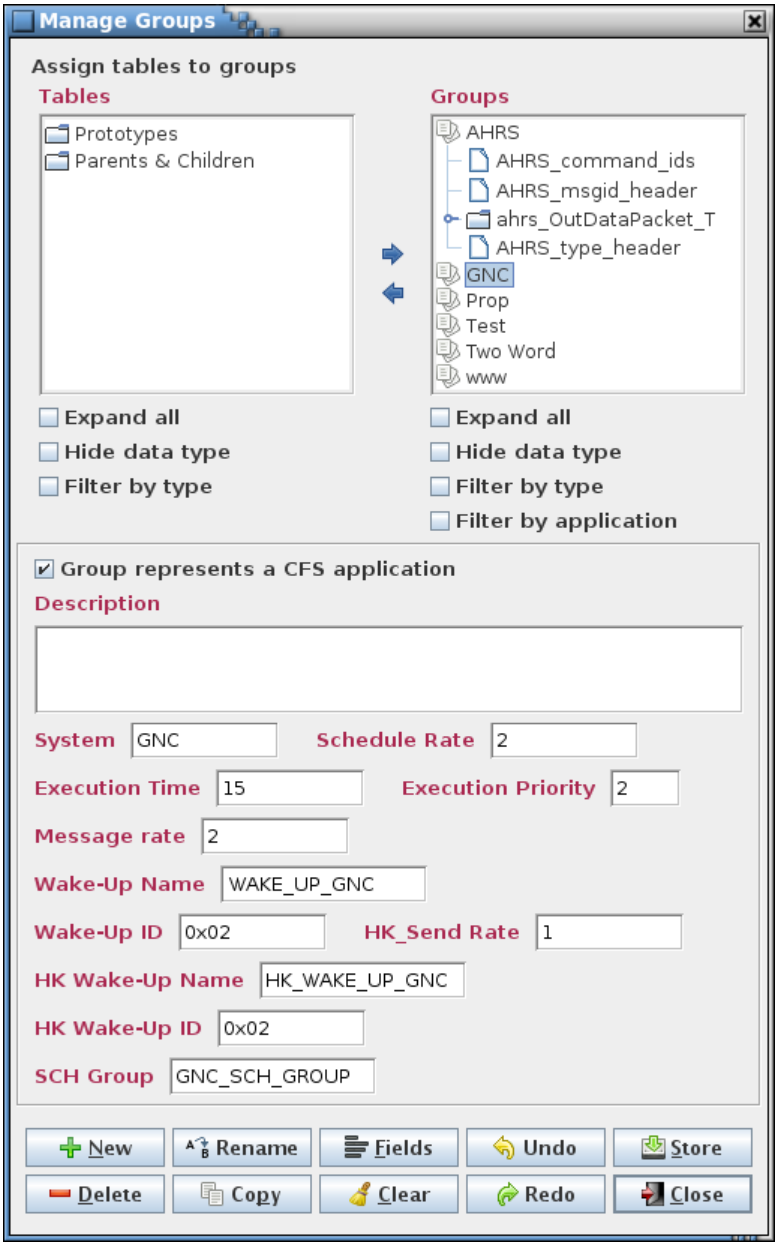


Figure 48. Manage Groups dialog

The upper left of the dialog contains a table tree (under the **Tables** heading). The upper right shows the groups and their trees (under the **Groups** heading). Both trees have the **Expand all** and **Filter by type** check boxes (see paragraph 4.5.3 for further details). The **Groups** tree has an additional filter, **Filter by application**, which causes the groups to be divided into two sub-trees, **Application** and **Other**. The **Application** sub-tree displays only those groups that represent a CFS application (see below for more details), and the **Other** sub-tree shows the remaining groups. In between the tree panes are arrows for moving tables in and out of the group(s). Each tree also has one or more check boxes, to

expand/collapse the tree and to filter the tree information. Below the trees is a check box for indicating that the group represents a CFS application and an input field for adding a description for the group. This description is used as text for a tool tip that appears in the table trees whenever the mouse pointer hovers over a group name.

The space separating the table and group trees delineates a split pane control that is used to resize these panels relative to one another. Position the mouse pointer between the two tree panels and when the pointer changes to a double-headed arrow press and hold the left mouse button. Space permitting, the adjoining panes can be resized by moving the mouse pointer left or right. Release the mouse button to exit resizing.

If a variable is selected in the **Tables** tree then every group to which it belongs is selected automatically in the **Groups** tree. If multiple tables are selected in the **Tables** tree then any selected groups are deselected. Selecting a non-grouped table deselects any highlighted group in the **Groups** tree.

To create a group select the **New** button and provide a group name and description in the input dialog that appears (Figure 49). The group name may not be blank, nor is the name allowed to match that of an existing group (including the automatically supplied group's name, 'All tables', which does not appear in the **Groups** tree). The group name may contain alphanumeric, spaces, and punctuation characters; there is no length constraint. If the **Group represents a CFS application** check box is selected then the group is automatically assigned a number of data fields appropriate for a CFS application (the fields may be altered using the **Fields** button after the group is created; a group's classification as a CFS application can be altered later – see below). These fields are **Schedule Rate**, **Execution Time**, **Execution Priority**, **Message Rate**, **Wake-Up Name**, **Wake-Up ID**, **HK Send Rate**, **HK Wake-Up Name**, **HK Wake-Up ID**, and **SCH Group**. The contents of these fields is used when populating the scheduler table created with the application scheduler (paragraph 4.9.4.3). If **Okay** is selected the new group's name appears in the group tree.

Figure 49. New Group dialog

To add tables to a group select the group in the **Groups** tree using the mouse or keyboard. Then, in the **Tables** tree, expand the tree as needed and select one or more tables using the mouse or keyboard. Multiple tables can be selected simultaneously by holding the Ctrl or Shift keys down when making a selection. Selecting a structure table automatically includes its child tables (and their children, etc.). Choosing a child table automatically includes its parent table, and its parent's parent, etc., up to the root table, but does not include any of its siblings (i.e., tables having the same parent and at the same tree level as the chosen table). Finally, select the right arrow button in the center of the dialog. The table(s) chosen appear in the selected group, and the group's tree is expanded to show the table(s) added. Note that the table hierarchy is preserved in the group's tree. More tables can be assigned to the group as described above.

To remove tables from a group expand the group's tree and select the table(s) to remove using the mouse or keyboard. Then select the left arrow button in the center of the dialog to delete the tables from the group. A table's children (and their children, etc.) are removed along with the chosen table.

To delete a group, first select it in the **Groups** tree, then select the **Delete** button. Multiple groups can be removed simultaneously if desired by highlighting them while using the Shift or Ctrl keys.

To rename a group, select a single group from the **Groups** tree, then press the **Rename** button. An input dialog appears with the name of the selected group in the input field. Alter the name as desired and select **Okay** to change the group's name. The renamed group name may not be blank, nor is the name allowed to match that of an existing group. Select **Cancel** to exit the input dialog without affecting the group's name.

To copy a group and its member tables, select a single group from the **Groups** tree, then press the **Copy** button. An input dialog appears with the name of the selected group in the input field with the text "_copy" appended. Alter the name as desired and select **Okay** to create a copy of the selected group. The copy has all of the tables assigned to it that are assigned to the original. The group name of the copy may not be blank, nor is the name allowed to match that of an existing group. Select **Cancel** to exit the input dialog without copying the group.

Data fields (see paragraph 4.6) may be assigned to a group. A group must first be selected in the **Groups** tree; this enables the **Fields** and **Clear** buttons. Select **Fields** to display the data field editor (see paragraph 4.6.1 for details on it use) for the currently selected group. Once a field is created it can have a value assigned, and the values for existing fields may be altered if desired. The data field values for the currently selected group can be cleared by selecting the **Clear** button.

A group's description can be added or changed by first selecting the group in the **Groups** tree. The current description for the group appears in the **Description** input field. The description can then be changed as desired.

When a group is selected a check box appears above the description field allowing the group's classification as a CFS application to be changed. If checked a number of default fields are automatically added below the description field, unless these fields are already present. Deselecting the check box does not remove these fields.

Changes made in the group manager (group additions or deletions, table assignments, data field updates, or changes to descriptions) are stored in the database only when the **Store** button is pressed. If changes have been made a confirmation dialog first appears. Select **Okay** to store the updates; select **Cancel** to exit the confirmation dialog without altering the database.

Select the **Close** button to exit the group manager dialog. If there are any unsaved group changes a dialog appears requesting confirmation to discard the changes. Select **Okay** to exit the group manager, losing any unsaved changes. Select **Cancel** to return to the group manager dialog.

The group manager button commands are summarized below:

New Create a new group.

Delete Delete the selected group(s).

Rename Rename the selected group.

Copy Create a copy of the selected group, including its member tables and data fields.

Fields Invokes the data field editor in order to create, alter, and delete data fields for the selected group.

Clear Replace the values in all data fields with blanks and deselect any check box data field for the selected group.

Undo Undoes the last action performed (table assignment, typing, paste, insert, delete, redo, etc.) on the selected group.

Redo Reverses the last action undone (table assignment, typing, paste, insert, delete, undo, etc.) on the selected group.

Store Stores the changes made to the groups in the group manager into the project database.

Close Closes the group manager window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the dialog.

4.9.3.10 Manage table types

The **Manage table types** command opens the table type editor (Figure 50). The editor window can be broken down into the following sections: the menu bar, table type tab(s), column definitions, type description, and buttons. An additional section, showing data fields, is visible below the description if data fields are created.

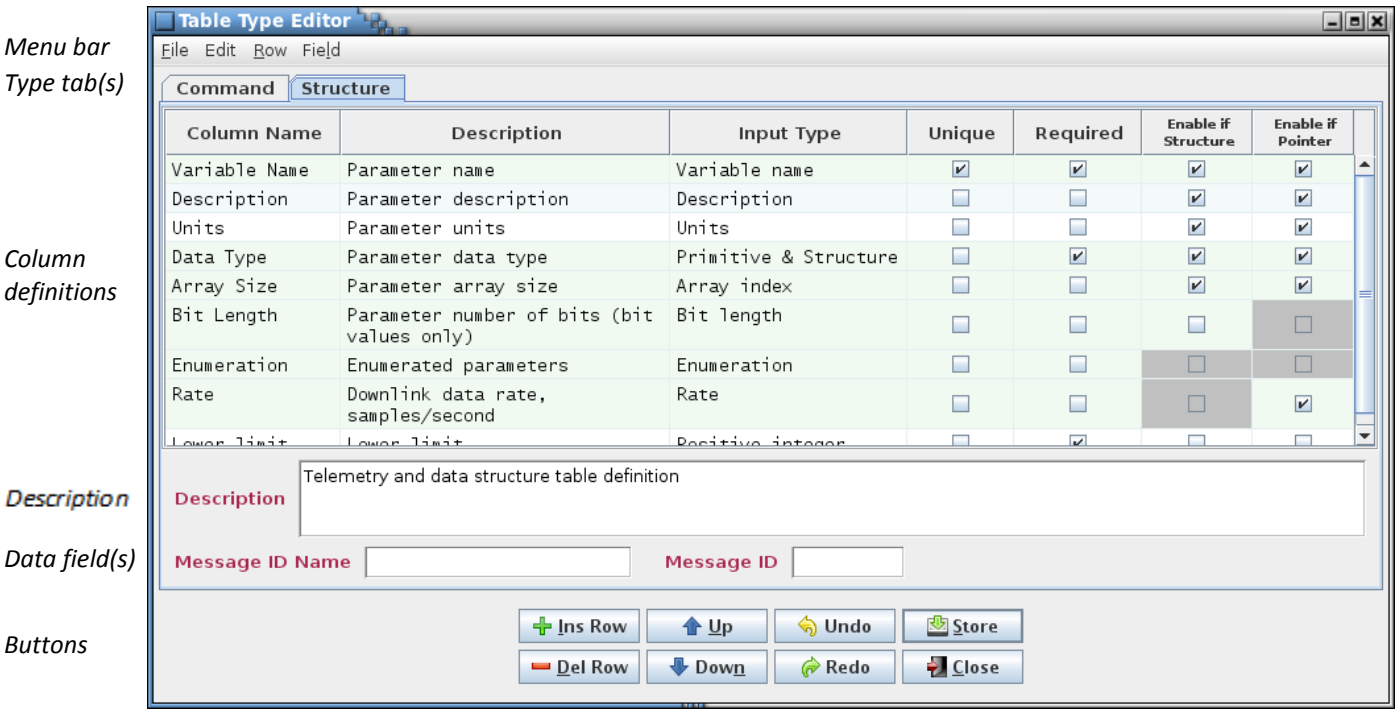


Figure 50. Table type editor

The menu bar has commands associated with the table type editor; the commands are described in subsequent paragraphs. The buttons, described in detail below, represent some of the more commonly used commands; each has a counterpart in the menu bar. Each type tab represents one of defined table types. As a default this includes the Structure and Command types. Any types created by the user also appear. The tabs are arranged in alphabetical order. Selecting the tab causes the editor to display the information for the selected table type.

Each row in the editor is a definition of a column that appears in each table of this type. The order of the column definitions determines the initial column order when the table is first displayed. The editor

columns can be sorted and repositioned as described in paragraph 4.4. Note that sorting the columns using the header does not affect the column order of tables created using the type. The displayed column order can be changed for each individual table in the table editor (see paragraph 4.9.3.2.4).

Editor cells that are grayed out cannot be changed. Selections in a columns definition can preclude other selections. For example, since only integer data types are valid for an enumeration, columns with an enumeration input type can't have a data type that's a structure and the **Enable if Structure** check box is disabled.

Column definition rows are highlighted when the all of the column definitions necessary to define a structure or command table are present and the column definition is one of these rows. In Figure 50 the column definitions for the variable name, data type, array size, bit length, enumeration, and rate are highlighted since these are the ones required to define a table representing a structure. If any one of these column definitions is removed then this highlighting is removed for all of the rows. The highlighting serves to indicate that the minimum column definitions are present to define the table type.

The editor column descriptions are as follows:

Column Name When a table of this type is displayed, this is the name that's displayed in the table's header for the column defined on this row of the editor.

Description This text is displayed as a tool tip whenever the mouse pointer hovers over this column's name in the table's header.

Input Type The input type constrains the type of value entered into the table cells for this column. If the value entered into the cell doesn't conform to the column's specified input type then a warning message dialog is displayed and the cell reverts to its previous value. The input types are selectable from the combo box pull-down menu that appears when a cell in the **Input Type** column is selected. See paragraph 4.7 for information on the available input types.

Unique This check box, if selected, indicates that each cell's data value must be unique within this column. If a duplicate value is entered into a cell then a warning message dialog is displayed and the cell reverts to its previous value.

Required This check box, if selected, indicates that the cell in this column requires a value. This causes the cell to be highlighted in yellow if it is empty. This does not force the user to populate the highlighted cell prior to saving changes to the table, but simply serves as a reminder that the information in this cell is considered important (for example, necessary to a script).

Enable if Structure This check box, if selected, indicates that the cells in this column allow input when the same row's data type column contains a structure reference. If the check box isn't selected then whenever a structure is selected as the data type the table's cell is grayed out and its value is blanked. *Note: This editor column only appears when the table type contains all column definitions necessary to represent a structure.*

Enable if Pointer This check box, if selected, indicates that the cells in this column allows input when the same row's data type column contains a pointer reference. If the check box isn't selected then whenever a pointer is selected as the data type the table's cell is grayed out and its value is blanked. *Note: This editor column only appears when the table type contains all column definitions necessary to represent a structure.*

The button commands mirror commands available in the editor menu bar and provide an easy method of accessing the commonly used editor commands. The button commands are described below:

Ins Row Inserts an empty row below the currently selected cell's row. If no cell is selected then the new row is inserted at the end of the table.

Del Row Deletes the row associated with each currently selected cell. If no row is selected then this has no effect.

Up Move the row(s) of the currently selected cell(s) up one row. This affects the order of the columns of new instances of this table type; it does not affect existing tables of this type. The displayed column order can be changed for each individual table in the table editor (see paragraph 4.9.3.2.4).

Down Move the row(s) of the currently selected cell(s) down one row. This affects the order of the columns of new instances of this table type; it does not affect existing tables of this type. Column order can be changed for each individual table in the table editor (see paragraph 4.9.3.2.4).

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.).

Store Stores the changes made to the currently displayed tab in the table type editor (not those in the other tabs) in the database. See paragraph 4.9.3.10.1.5 for further details.

Close Closes the table type editor window. If any changes for any of the tabs have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

The commands in the editor menu bar are described in the following paragraphs.

4.9.3.10.1 File menu

4.9.3.10.1.1 New type

The **New type** command allows the user to create a new table type. This command is enabled only for a user with read/write or administrator access. A dialog appears with an input field for entering the new type's name (Figure 51). The three radio buttons allow automatically populating the new type with those columns that are required for a structure or command table type; selecting **None** results in a table type with no columns defined. Select **Create** to create the new type, which is opened in the table type editor. The editor can then be used to populate the table type with column definitions and edit any that were automatically added, and afterwards new tables of this type may be created and edited. Select **Cancel** to exit the dialog without creating a new table type.

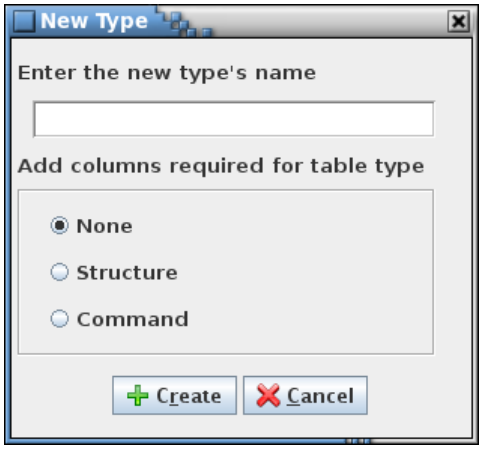


Figure 51. New table type dialog

4.9.3.10.1.2 Copy type

The **Copy type** command is used to create a new table type from an existing one, including all of its column definitions and default data fields. This command is enabled only for a user with read/write or administrator access. The active tab in the table type editor determines which type is to be copied, so the intended tab must be selected prior to executing the copy command. A dialog appears with an input field for entering the name of the table type's copy. The name of the selected type is displayed with "_copy" appended. After altering the name as desired, select **Copy** to create a copy of the type. Select **Cancel** to exit the dialog without creating a copy.

4.9.3.10.1.3 Rename type

The **Rename type** command is used to rename an existing table type. This command is enabled only for a user with read/write or administrator access. The active tab in the table type editor determines which type is to be renamed, so the intended tab must be selected prior to executing the rename command. A dialog appears with an input field for entering the new name for the table type. The name of the selected type is automatically displayed. After altering the name as desired, select **Rename** to rename the table type. All tables of the renamed type are changed to reference the new table type name. Select **Cancel** to exit the dialog without renaming the table type.

4.9.3.10.1.4 Delete type

The **Delete type** command deletes an existing table type. This command is enabled only for a user with read/write or administrator access. The active tab in the table type editor determines which type is to be deleted, so the intended tab must be selected prior to executing the delete command. A confirmation dialog appears. Selecting **Delete** removes the table type *and all tables of the deleted type from the project database*. Select **Cancel** to exit the dialog without deleting the table type or any tables.

4.9.3.10.1.5 Store current

The **Store current** command performs the identical action to the **Store** button. This command is enabled only for a user with read/write or administrator access. The command stores the changes made to the currently displayed tab in the table type editor (not those in the other tabs) in the project database. Afterwards, any table created using this table type inherits the type's columns and data fields.

All existing tables of this type, including those in any open table editors, are updated immediately with the column and data field changes. New data fields are added to existing tables; however, deleted data fields are not removed from existing tables. Changes to data field values are applied based on the **Overwrite values** check box described in paragraph 4.9.3.10.4.3.

A confirmation dialog appears allowing the user to choose between continuing with the store operation and canceling it.

4.9.3.10.1.6 Store all

The **Store all** command is similar to the **Store current** command described above, except that it stores the changes made to all the table type editor tabs in the project database. This command is enabled only for a user with read/write or administrator access. All existing tables of the affected type(s), including those in any open table editors, are updated immediately with the changes. A confirmation dialog appears allowing the user to choose between continuing with the store operation and canceling it.

4.9.3.10.1.7 Print current

The **Print current** command prints the contents of the currently displayed tab to be sent to a printer. A dialog first appears allowing the user to select the printer (or file) and adjust the page setup. Selecting

Print causes the editor contents, including the data fields (if any), to be output to the selected printer (or file). Selecting **Cancel** removes the print dialog without printing the table type editor contents.

4.9.3.10.1.8 Search

Selecting the **Search** command causes a search dialog to be displayed. The behavior of this dialog is identical to that for the table editor dialog except that the search is confined to the currently displayed table type editor. See paragraph 4.9.3.2.1.8 for details on use of the search dialog.

4.9.3.10.1.9 Close

The **Close** command performs the identical action to the **Close** button. The command closes the table type editor window. If any changes for any of the tabs have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.3.10.2 Edit menu

4.9.3.10.2.1 Copy

The **Copy** command places the contents of the highlighted cell(s) into the operating system's clipboard. This information can then be pasted into another cell or input field in the application, or into applications other than CCDD. The Ctrl-C keys perform the same operation.

4.9.3.10.2.2 Paste

The **Paste** command places the contents of the operating system's clipboard into the editor. The paste location is determined by the leftmost and uppermost highlighted cell. The rows and columns of the copied cells are placed into the editor beginning at this location and extending down and to the right, overwriting the existing data in the cells. If insufficient columns exist for the pasted data then the excess column information is ignored. Extra rows are inserted at the bottom of the table to provide room for data that would be placed below the editor's last row. See paragraph 4.9.3.10.2.3 on inserting copied data without overwriting the existing cell contents. The Ctrl-V keys perform the same operation.

4.9.3.10.2.3 Insert

The **Insert** command behaves similarly to the **Paste** command (paragraph 4.9.3.10.2.2) except that no editor data is overwritten. Instead, rows are inserted, beginning at the row below the upper- and leftmost highlighted cell, to accommodate the pasted values. The Ctrl-I keys perform the same operation.

4.9.3.10.2.4 Undo

The **Undo** command performs the same action as the **Undo** button. The command undoes the last action performed (typing, paste, insert, delete, redo, etc.) in the currently displayed table type editor. The Ctrl-Z keys perform the same operation.

4.9.3.10.2.5 Redo

The **Redo** command performs the same action as the **Redo** button. The command reverses the last action undone (typing, paste, insert, delete, redo, etc.) in the currently displayed table type editor. The Ctrl-Y keys perform the same operation.

4.9.3.10.2.6 Clear data

The **Clear data** command empties all of the currently displayed editor's cells.

4.9.3.10.2.7 Add command arguments

The **Add command arguments** command appends a default set of command argument column definitions to the currently displayed table type. The command is only enabled if the table type represents a command. The index for the added arguments is determined by scanning the existing argument column names for the pattern "Arg # ", where '#' is the argument index (initialized to 1). If a column is found matching the pattern then the index is incremented and the process is repeated. This continues until no match is found; the new arguments are named using the index value.

4.9.3.10.3 Row menu

4.9.3.10.3.1 Insert row

The **Insert row** command performs the same action as the **Ins Row** button. The command causes an empty row to be inserted below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table. The Insert key performs the same operation.

4.9.3.10.3.2 Delete row(s)

The **Delete row(s)** command performs the same action as the **Del Row** button. This command deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this command has no effect. The Delete key performs the same operation.

4.9.3.10.3.3 Move up

The **Move up** command performs the same action as the **Up** button. This command causes the row(s) of the currently selected cell(s) to move up one row relative to the remaining rows. Since each row is a column definition, this affects the order of the columns of new instances of this table type; it does not affect existing tables of this type. Column order can be changed for each individual table in the table editor (see paragraphs 4.9.3.2 and 4.9.3.2.4).

4.9.3.10.3.4 Move down

The **Move down** command performs the same action as the **Down** button. This causes the row(s) of the currently selected cell(s) to move down one row relative to the remaining rows. Since each row is a column definition, this affects the order of the columns of new instances of this table type; it does not affect existing tables of this type. Column order can be changed for each individual table in the table editor (see paragraphs 4.9.3.2 and 4.9.3.2.4).

4.9.3.10.4 Field menu

4.9.3.10.4.1 Manage fields

The **Manage fields** command allows the user to create, alter, and delete default data fields for the type represented by the active table type editor tab. See paragraph 4.6 for information regarding data fields and use of the data field editor.

The fields manipulated by the field editor are displayed below the table type editor table and description when the **Update** button is pressed. The table type editor's **Store** button or command must be used to store the changes in the database and apply them to the tables. The structure table data field editor allows the user to assign fields to only parent or child structure tables, or to all structure tables.

After the field editor is closed values can be entered into the data fields; these become default values for the fields in the tables to which the fields are applied. When the field updates are stored all tables of

the affected table type are updated, including those in any open table editors, and tables of this type that are subsequently created have the default fields. If an existing table already has a field of the same name then it is not added; however, the field size, data type, required status, description, and value (depending on the state of the **Overwrite values** check box; see paragraph 4.9.3.10.4.3) are updated in the table to match the default.

Fields can only be added to tables using this method. If a default field's name is changed then this is considered a new field and is added to the tables of the affected type if the type is stored. If a default field is deleted then there is no effect on the tables when the type is stored.

4.9.3.10.4.2 Clear values

The **Clear values** command clears the contents of all of the currently displayed editor's data fields. A confirmation dialog is first displayed. Selecting **Okay** causes all of the data field values to be blanked. Selecting **Cancel** exits the dialog without affected the data field values.

4.9.3.10.4.3 Overwrite values

The **Overwrite values** command determines how the default field values are applied when the table type editor **Store** button is selected. The default setting is unchecked.

If the **Overwrite values** check box is not selected then the data field value changes are not applied to existing tables of this table type if the tables already contain the affected data field. If an existing table does not already have the data field then the field is added with the default value, regardless of the check box status.

If the **Overwrite values** check box is selected then all existing tables of the updated table type that already contain the data field have the contents of that field replaced with the value in the type editor data field.

4.9.3.11 Manage data types

The Data Type Editor (Figure 52) provides a means of creating, modifying, and deleting primitive data type definitions (see paragraph 4.5.4 for more information on data types). When a project database is first created the primitive data types default to those shown in Table 5.

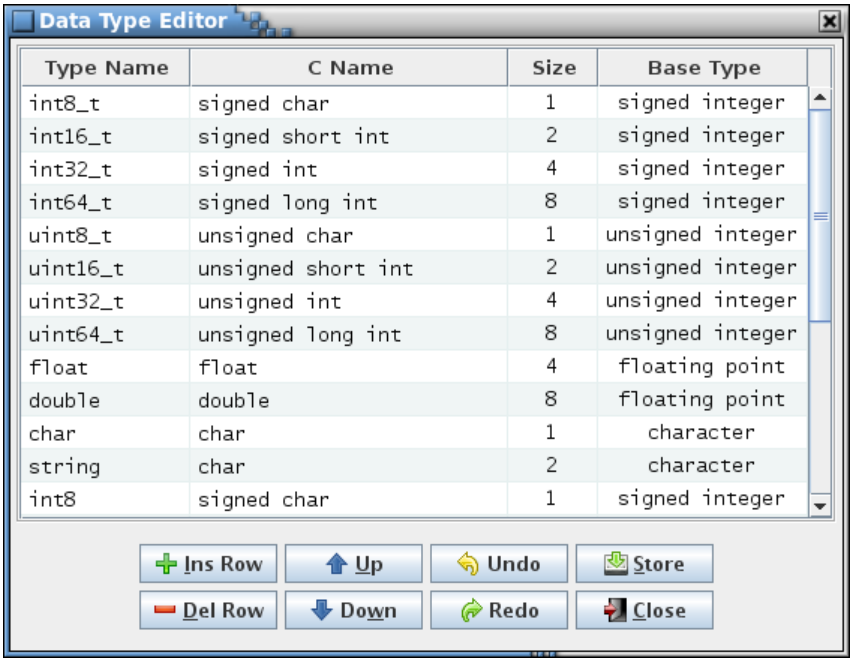


Figure 52. Data Type Editor dialog

The editor column descriptions are as follows:

Type Name The type name is the text that represents the data type in a data table cell. Data type names must adhere to C-language naming conventions; i.e., begin with an alphabetic or underscore character, followed by alphabetic, numeric, or underscore characters. The data type names are case insensitive and must be unique. If the type name is left blank then the text in the C Name column determines the data type name displayed in the data table cell.

C Name The C name is the C-language equivalent of the data type and may contain spaces. It is available to scripts and web applications (for example, a script can create a header file of `typedef` statements using the type name and C name combinations). One or more trailing asterisks are allowed if the corresponding base type is 'pointer'. The C name is used as the data type in a data table cell if the corresponding type name is blank.

Size Size, in bytes, occupied by this data type. The size must be an integer greater than 0.

Base Type The data type's base type: signed integer, unsigned integer, floating point, character, or pointer. The base type and the size determine how the data type is handled by the application.

Each row in the table is a data type definition. The **Type Name** or **C Name** columns determine the data type displayed in the data type column drop down menus. At least one of these columns must contain text. The type name is used if it isn't blank. If the type name is blank then the C type name is used as the data type name. Valid values must be entered in the **Size** and **Base Type** columns for every definition. The rows can be sorted by selecting the column headers, as with other table editors in the application.

If a pointer base type is selected then an asterisk (*) is automatically appended to the C name (unless the cell is empty). Conversely, if the base type is changed from a pointer to something else then any trailing asterisk(s) in the C name is automatically removed. A pointer to a pointer (or a pointer to a pointer to a pointer, etc.) can be indicated by appending the requisite number of asterisks. Below is an example of creating a pointer to a structure named **myStruct**:

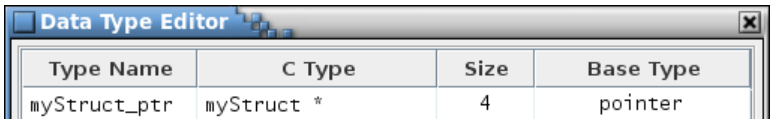


Figure 53. Example pointer to a structure data type

As an aid to creating a data type that represents a pointer to a structure a pop-up list of alphabetically arranged prototype structure table names can be displayed from which a structure name can be selected. This pop-up is displayed by pressing Ctrl-S and is only available when editing a cell in the **Type Name** or **C Type** column and if the **Base Type** column for the edited row is blank or a pointer. Use the mouse or keyboard to highlight the structure name to insert. Once the desired structure name is highlighted either press the left mouse button or the Enter key. The structure name is inserted into the table cell, replacing any selected text (Figure 54). Press the Escape key to remove the structure name pop up dialog without inserting a structure name.

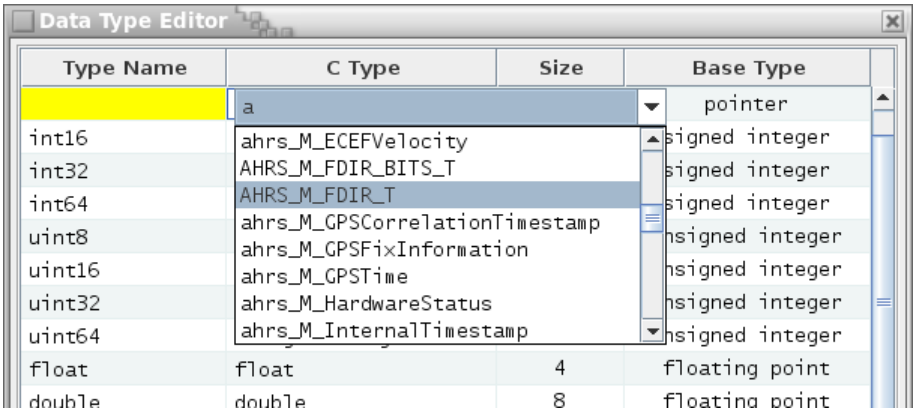


Figure 54. Structure name pop-up

If a data type is currently in use in a data table then the size and base type may be constrained by the values in other columns on the same row of the affected table. For example, if a data type is a 2-byte integer and is used in a data table where the parameter is assigned a bit length of 10 bits then the data type size can't be changed to a single byte since a single byte's 8 bits is insufficient for the 10-bit parameter. The instance where the bit length exceeds the desired size must first be altered before the size can be updated. If an invalid size or base type is entered a dialog appears indicating the tables where the inconsistency exists, and the table cell reverts to its previous value.

The button commands are described below:

Ins Row Inserts an empty row below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table.

Del Row Deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this has no effect. A data type cannot be deleted if it is currently used in a table; all references must be removed before the data type can be deleted.

Up Move the row(s) of the currently selected cell(s) up one row relative to the remaining rows. The order of the data type definitions in the editor has no effect on data type usage, though it does determine the order of the types in the data type combo box lists. The capability to arrange the rows is solely for the user to group the data types as desired.

Down Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the data type definitions in the editor has no effect on data type usage, though it does determine the order of the types in the data type combo box lists. The capability to arrange the rows is solely for the user to group the data types as desired.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.).

Store Stores the changes made to data type definitions in the data type editor into the project database. All tables are updated with the changes, including any tables currently open in a table editor.

Close Closes the data type editor window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.3.12 Manage input types

The Input Type Editor (Figure 55) provides a means of creating, modifying, and deleting custom input type definitions (see paragraph 4.7 for more information on input types). These input types can then be applied to data table columns via the table type editor (see paragraph 4.9.3.10) and to data fields via the data field editor (see paragraph 4.6.1), the same as with the default input types.

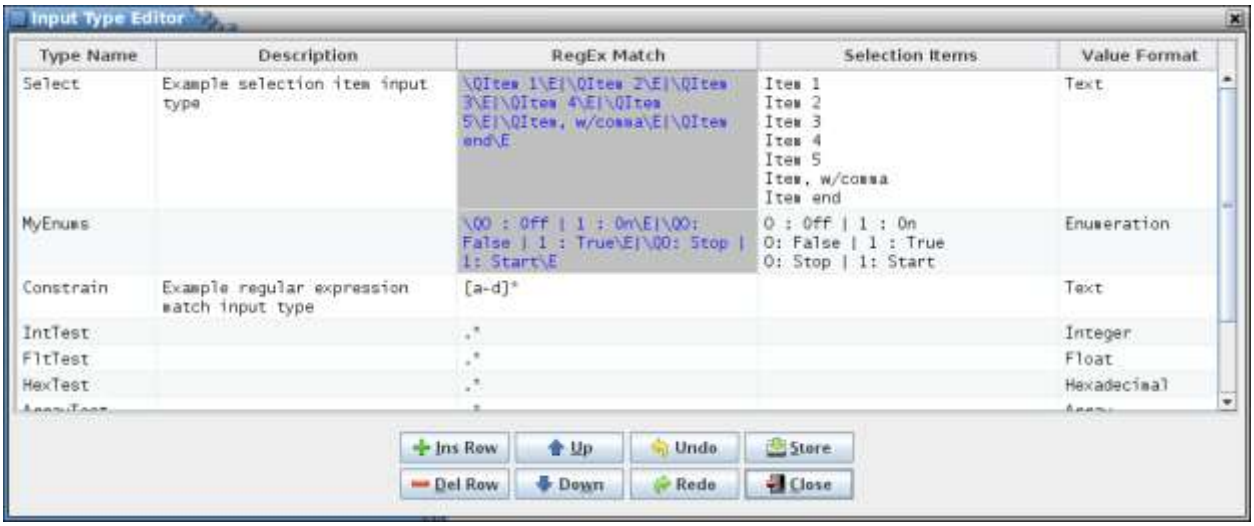


Figure 55. Input Type Editor dialog

The editor column descriptions are as follows:

Type Name The input type name is the text that appears in the table type and data field editors' **Input Type** column drop down menus. Input type names are case insensitive and must be unique (this includes the default input type names in paragraph 4.7).

Description The input type's description.

RegEx Match This is the regular expression used to constrain the values entered into a data table cell of data field of this input type. Editing of the **RegEx Match** column is disabled if the **Selection Items** cell isn't blank.

Selection Items If the data table cell or data field is constrained to specific text strings then these strings can be entered into this column. The items must be separated by line feed characters (use **Alt-Enter** to insert a line feed). Data table cells and data fields using this input type display a drop down menu displaying these selection items from which to choose. If the **Selection Items** column isn't empty then editing of the **Regex Match** column is disabled. The **RegEx Match** is automatically populated with the regular expression that constrains the input type to the selection items and the **Value Format** selections are altered. Deleting the contents of the **Selection Items** cell restores the ability to edit the **Regex Match** and **Value Format** columns, but leaves their contents untouched.

Value Format When selected, a drop down menu appears displaying various format options. The options available depend on whether or not the **Selection Items** column is empty. If empty, the value format applies the specified formatting to a value entered in a data table cell or data field. The following describes the result of applying the value formats

to the entered text (note that if the format isn't applicable to the values allowed by the input type's regular expression then the format is ignored and the text is unchanged):

Text No formatting is applied; the text remains unchanged. This is the default format.

Array This format expects one or more numerals, separated commas. The spaces (if any) and commas between the numbers are replaced by ", ". This is the same formatting used for array sizes.

Boolean The table cell or data field is treated as a boolean and is displayed as a check box. The **RegEx Match** column value is changed to the regular expression matching a boolean value.

Float The text is formatted as a floating point value in the form #.#. Leading zeroes and excess trailing zeroes in the decimal portion are removed.

Hexadecimal The text is formatted as a hexadecimal number. "0x" is prepended if not already present.

Integer The text is formatted as an integer value. Leading zeroes and decimal values are removed.

Number The text is formatted as a floating point value in the form #.#. If the value is an integer then the decimal and trailing zero is removed.

If the **Selection Items** column isn't empty the the **Value Format** options are changed to the ones shown below. These options do not reformat the selection items; instead they define how the cells or fields using this input type are handled by the program, such as for the script data access methods, or when parsing table data during EDS and XTCE XML exports and imports. For example, the script data access methods that return enumeration column values will consider a column with the **Enumeration** value format as an enumeration column.

Text No special treatment is performed for the selection items. This is the default format.

Enumeration The selection items are treated as if the input type is an enumeration. This is useful if the project requires a number of common enumerations that are used frequently throughout the tables. This input type allows the user to quickly choose from the acceptable enumerations.

Minimum The selection items are treated as if the input type is a minimum value. In a maximum value column is present then the value in this column selected from the list must be less than or equal to the maximum value.

Maximum The selection items are treated as if the input type is a maximum value. In a minimum value column is present then the value in this column selected from the list must be greater than or equal to the minimum value.

Each row in the table is an input type definition. The rows can be sorted by selecting the column headers, as with other table editors in the application. Every definition requires a value in the **Type Name**, **RegEx Match**, and **Value Format** columns; the remaining column may be blank. If an input type is currently in use in a data table or data field then the input type cannot be deleted; if attempted a dialog appears indicating the tables and fields using the input type. If the regular expression or selection item names are changed then a check is made against all of the tables and fields using the input type. If the table cell or data field value no longer conforms to the regular expression then the value in the cell or field is blanked.

The button commands are described below:

Ins Row Inserts an empty row below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table.

Del Row Deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this has no effect. An input type cannot be deleted if it is currently used in a table; all references must be removed before the input type can be deleted.

Up Move the row(s) of the currently selected cell(s) up one row relative to the remaining rows. The order of the input type definitions in the editor has no effect on input type usage. The capability to arrange the rows is solely for the user to group the input types as desired.

Down Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the input type definitions in the editor has no effect on input type usage. The capability to arrange the rows is solely for the user to group the input types as desired.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.).

Store Stores the changes made to input type names, descriptions, regular expression match strings, format types, or selection items in the input type editor into the database. All tables are updated with the changes, including any tables currently open in a table editor.

Close Closes the input type editor window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.3.13 Manage macros

The Macro Editor (Figure 56) provides a means of creating, modifying, and deleting macro definitions (see paragraph 4.5.7 for more information on macros).

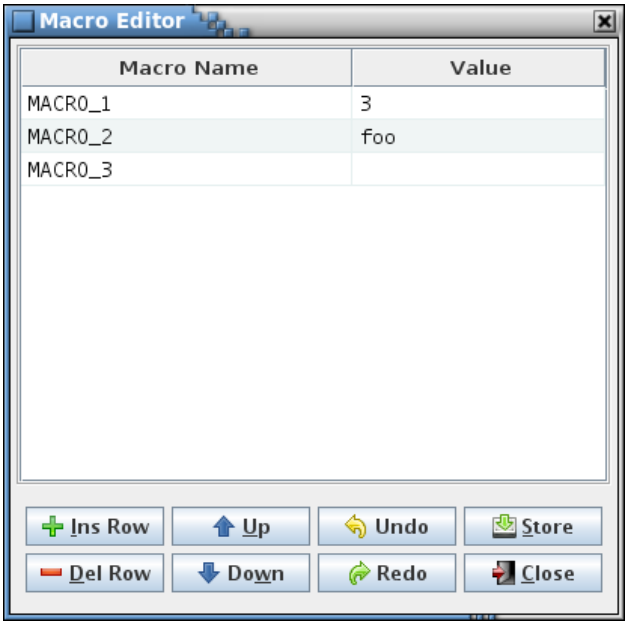


Figure 56. Macro Editor dialog

The editor column descriptions are as follows:

Macro Name The macro name is the text that represents the macro's value in a data table or macro editor cell (the macro name is delimited by pairs of '#' characters and highlighted in the cell). Macro names are case insensitive and must be unique.

Value The macro value is the text that the macro name represents.

Each row in the table is a macro definition. The rows can be sorted by selecting the column headers, as with other table editors in the application. Every definition requires a name, but the value may be blank. If a macro is currently in use in a table then the macro value is constrained by the input type of the column(s) in which the macro is referenced. For example, if a macro is inserted into a column of input type "Array index" then the macro's value can be blank or must evaluate to a number (or a series of numbers separated by commas), as required by the array index input type. If an invalid value is entered a dialog appears indicating the tables where the inconsistency exists, and the editor cell reverts to its previous value.

Macro values can reference other macros. Any macros referenced in a macro value are highlighted. Each macro in the macro's value, when the macro is expanded, is replaced by its value. Circular macro references are not allowed (i.e., a macro references itself in its value); if detected a warning dialog is displayed and the cell contents reverts to its previous value.

Macro values can also use the *sizeof(data type)* call. Any *sizeof()* calls in the macro value are highlighted similar to macro references. When expanded the *sizeof()* call is replaced by the size in bytes of the specified primitive or structure data type.

While editing a macro value cell, the Ctrl-M keys can be used to display a pop-up list of macros for insertion into the value. Position the text cursor or highlight one or more characters to be replaced, then press Ctrl-M. The pop-up list appears as shown in Figure 57.

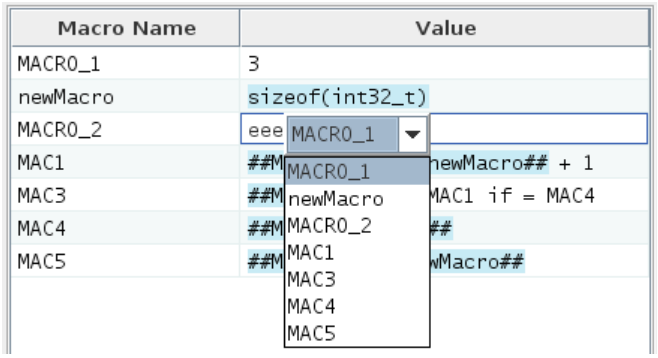


Figure 57. Example of macro name selection pop-up dialog in a macro value cell

Pressing Ctrl-Shift-M temporarily replaces every macro with its corresponding value in the macro editor **Values** column. Releasing the Ctrl-Shift-M keys restores the macro names in the cells.

The button commands are described below:

Ins Row Inserts an empty row below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table.

Del Row Deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this has no effect. A macro cannot be deleted if it is currently used in a table; all references must be removed before the macro can be deleted.

Up Move the row(s) of the currently selected cell(s) up one row relative to the remaining rows. The order of the macro definitions in the editor has no effect on macro usage. The capability to arrange the rows is solely for the user to group the macros as desired.

Down Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the macro definitions in the editor has no effect on macro usage. The capability to arrange the rows is solely for the user to group the macros as desired.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.).

Store Stores the changes made to macro names or values in the macro editor into the database. All tables are updated with the changes, including any tables currently open in a table editor.

Close Closes the macro editor window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.3.14 Message IDs

Message IDs is a sub-menu of commands relating to message ID names and numbers, described in the following paragraphs.

4.9.3.14.1 Assign IDs

The **Assign IDs** command provides a method for automatically assigning a unique message ID number to table cells and data fields having an input type of 'Message ID' (see paragraph 4.7). This command is enabled only for a user with read/write or administrator access. Telemetry message IDs are assigned in the telemetry scheduler (see paragraph 4.9.4.2). Automatic assignment can be limited to structure

tables, command tables, other type tables, and groups as desired. A dialog appears (Figure 58) containing a tab for structure tables, command tables, other table types, and groups.

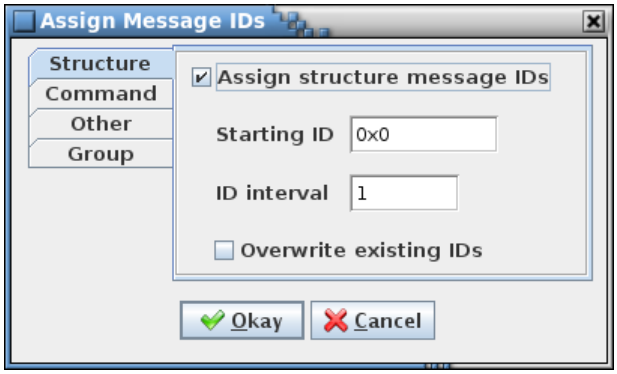


Figure 58. Assign Message IDs dialog

For each tab there are two check boxes and two input fields. The first check box is used to determine if the message IDs associated with the selected tab are to be updated. For the table type tabs this affects both table cells and data fields, and for the Group tab this affects only group data fields. If this check box is unchecked the remaining fields are disabled and ignored. The first input field is the starting ID number, in hexadecimal (the '0x' preceding the number is optional). The second field is the ID interval which is used to calculate the next ID value in the sequence - the default is 1; any positive integer value is valid. The final check box determines whether or not existing table/group message ID numbers are updated or left as is. When checked, existing message IDs are replaced unless the ID is flagged as protected. Message IDs are protected if a '#' character is appended to the ID value (example: ox1234#).

When **Okay** is selected then the tables representing a structure, command, or other type, and group data fields (depending on whether or not the associated check box is selected) are checked to determine if they have a column(s) or data field(s) with the input type of 'Message ID.' If so, then the column/field value is assigned a message ID number. IDs are assigned beginning with the starting ID number, and with each subsequent ID number equal to the previous number plus the interval value. Table column message IDs are assigned (for all tables) before message ID data fields. Macros are allowed in the table columns representing message IDs; however, the auto-assignment process will overwrite the macros with message ID values if the overwrite check box is selected for the table's type. An ID is skipped if it is listed in the reserved message ID list (see 4.9.3.14.2), assigned to table columns and/or data fields for a table type that doesn't have the overwrite check box selected, assigned to a group data field (unless the Group overwrite check box is selected), or assigned to telemetry messages in the telemetry scheduler (see paragraph 4.9.4.2). If the overwrite check box isn't selected then the values for any existing structure (command, other, group) message ID data fields are also skipped in order to avoid duplicate ID values. This action also updates the project database and the message ID number columns and/or data fields for any open table editors. Select **Cancel** to exit the dialog without altering the message ID values.

4.9.3.14.2 Reserve IDs

The editor dialog shown in Figure 59 to appears when the **Reserve message IDs** command is selected. This dialog allows message IDs, either singly or as ranges, to be flagged as reserved. This means that when automatic assignment of message IDs is performed (see paragraphs 4.9.3.14.1 and 4.9.4.2) the message IDs in the reserved table are skipped.

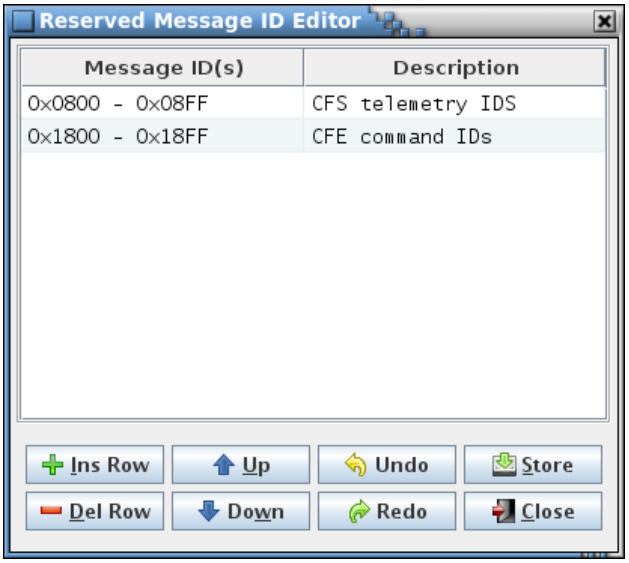


Figure 59. Reserved Message ID Editor dialog

The editor column descriptions are as follows:

Message ID(s) This column can contain a single hexadecimal number, optionally prepended with "0x", or a range of IDs consisting of two hexadecimal numbers separated by a hyphen (-). If a range is entered the second number must be greater than the first. Message IDs are not allowed to be duplicated in the table. This includes IDs falling within an existing range or overlapping of two ranges.

Description This column can be used to describe the ID or ID range. It may remain empty.

Each row in the table is a reserved message ID definition. Every definition requires a message ID or ID range, but the description may be blank. If an invalid or duplicate value is entered a dialog appears indicating the problem, and the editor cell reverts to its previous value. The rows can be sorted by selecting the column headers, as with other table editors in the application.

The button commands are described below:

Ins Row Inserts an empty row below the currently selected cell's row. If cells in multiple rows are selected then the new row is inserted below the lowest one. If no cell is selected then the new row is inserted at the end of the table.

Del Row Deletes the row associated with each currently selected cell. If cells in multiple rows are selected then each of the rows is deleted. If no row is selected then this has no effect.

Up Move the row(s) of the currently selected cell(s) up one row relative to the remaining rows. The order of the reserved message ID definitions in the editor has no effect on ID usage. The capability to arrange the rows is solely for the user to group the IDs as desired.

Down Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the reserved message ID definitions in the editor has no effect on ID usage. The capability to arrange the rows is solely for the user to group the IDs as desired.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.).

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.).

Store Stores the changes made to reserved message IDs or description in the reserved message ID editor into the project database.

Close Closes the reserved message ID editor window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.3.14.3 Show all IDs

The **Show all IDs** command displays a table showing all of the message IDs in a project along with their corresponding message names and the entity (table or group) in which the message name and ID are found. Figure 60 is an example of the table produced by the command. For a table cell or data field containing a message name and ID to be recognized as such it must have the input type **Message name & ID** (see paragraph 4.7 for more information on input types).

A script data access method, *getMessageOwnersIDsAndNames*, is provided that returns the same information as is produced by this command; see the reference in Table 10 for details.

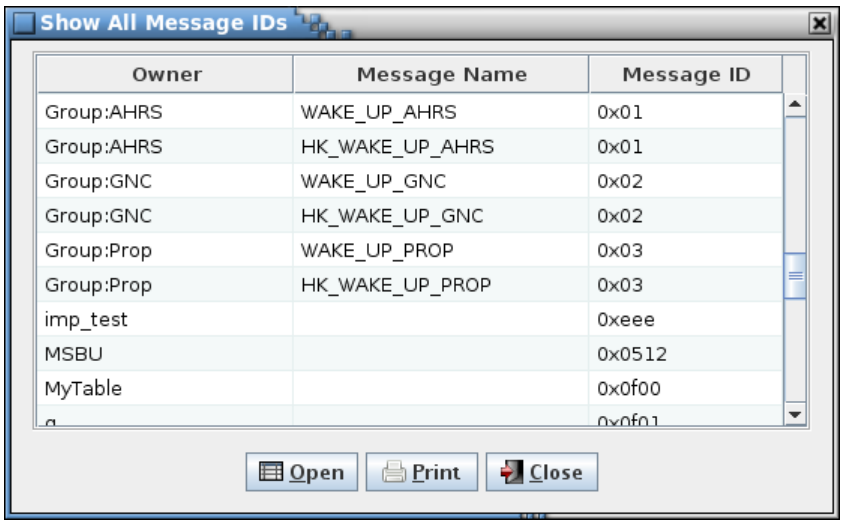


Figure 60. Example Show all IDS dialog

If one or more cells is selected and the **Open** button is pressed then the table(s) associated with the selected cell(s) are opened in a table editor. A row is ignored if it contains a message ID belonging to a group or telemetry message.

Selecting the **Print** button opens a printer selection dialog in order to print a copy of the table to the selected printer or file. Selecting **Close** closes the message ID dialog.

4.9.3.14.4 Find duplicates

The **Find duplicates** command searches the project database table cells, data fields, and telemetry messages for message ID values used more than once, and displays a dialog showing these IDs and where they are located (see the example dialog in Figure 61). If no duplicate message ID exists the dialog's table is empty.

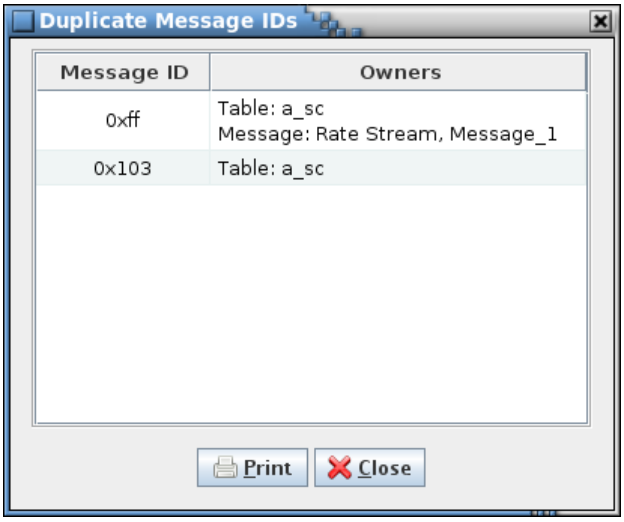


Figure 61. Example Duplicate Message IDs dialog

The **Message ID** column shows the duplicated ID as a hexadecimal value; the table is sorted based on the ID value. The **Owners** column displays the location(s) where the message ID is referenced – this can be a table cell or data field (table path and name is preceded by "Table:"), or telemetry message (message data stream and message name is preceded by "Message:"). The rows can be sorted by selecting the column headers, as with other tables in the application. Column order can be changed by dragging a column to a new position.

Selecting **Print** causes the dialog contents to be output to the selected printer (or file). Selecting **Close** closes the duplicate message IDs dialog.

4.9.3.15 Manage project fields

The **Manage project fields** command causes the project data field manager dialog, shown in Figure 62, to appear. Data fields (see paragraph 4.6) may be associated with a project, similar to how they can be associated with specific data tables and groups. The project's description can also be edited via this dialog (the description can also be edited using the **Rename** command in the **Project** menu; see paragraph 4.9.2.4).

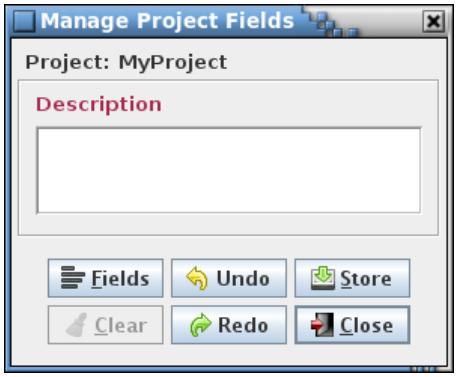


Figure 62. Project data field management dialog

The **Description** field appears by default and cannot be deleted. This field is enabled only for a user with administrator access. The **Fields** button causes the data field editor (see paragraph 4.6.1 for details on it use) to appear; data fields can be added, modified, or removed via this editor.

The project data field manager button commands are summarized below:

Fields Invokes the data field editor in order to create, alter, and delete data fields for the project.

Clear Replace the values in all data fields with blanks and deselect any check box data field for the project.

Undo Undoes the last action performed (typing, paste, insert, delete, redo, etc.) in the manager dialog.

Redo Reverses the last action undone (typing, paste, insert, delete, undo, etc.) in the manager dialog.

Store Stores the changes made to the project description and data fields in the manager dialog into the project database.

Close Closes the project data field manager window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the dialog.

4.9.3.16 Show/edit fields

The purpose of the **Show/edit fields** command is to provide a means of displaying, editing, and removing data fields for one or more data tables and/or groups via a single editor (as opposed to displaying the data fields for a specific owner table in a table editor or group in the group manager). Selecting the **Show/edit fields** command produces a dialog displaying a table tree and a set of check boxes, one for each unique data field name currently in use by the project's data tables. See Figure 63 for an example; if no tables exist or no data fields are currently assigned then a warning dialog appears instead indicating there is nothing to display.

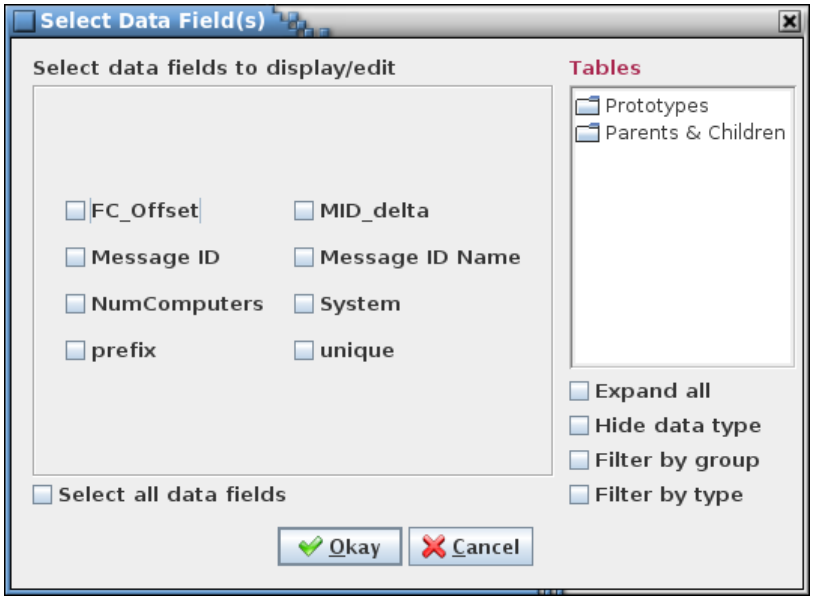


Figure 63. Example Select Data Field(s) dialog

The user chooses the field(s) to display/edit by selecting the field's associated check box. The **Select all data fields** check box is used to alternately select and deselect all of the data field check boxes. The fields can be filtered by selecting one or more tables from the table tree – only the selected fields in the selected tables are displayed in the editor. Selection of a header node in the table tree (e.g., 'Parents & Children', or a group name if group filtering is enabled) selects all tables under that header. If no tables are selected then no filtering occurs and the selected data field(s) are displayed for any table. Selecting

the **Okay** button opens the data field editor, while the **Cancel** button closes the dialog without opening the editor. An example of the editor dialog that appears is shown in Figure 64.

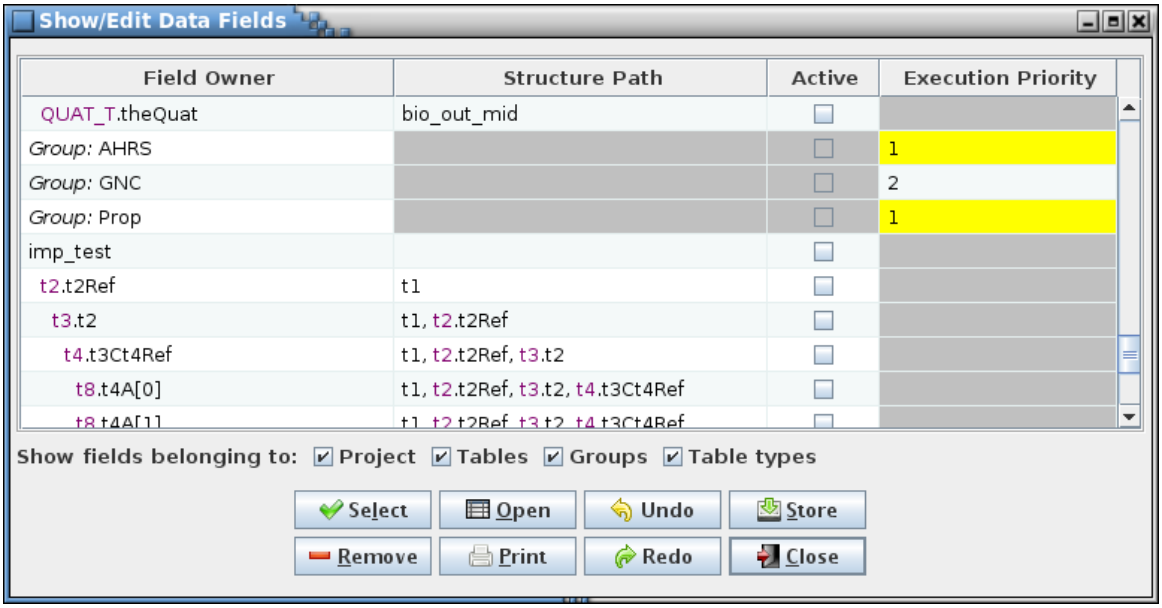


Figure 64. Example Show/Edit Data Fields dialog

The first column, **Field Owner**, displays the data field owner. For a top-level structure table or non-structure table this is the table's name. For a child table the child's prototype and instance (or variable) name are displayed in the format *prototype.instance*, and are indented by an amount based on the number of levels the child is from its root structure. Project-level fields display "*Project:*" as the field owner. Fields that belong to a group display the group's name with "*Group:*" prepended, and default data fields (those belonging to a table type definition) display the table type name prepended with "*Type:*".

The second column, **Structure Path**, displays the structure path for child tables, listing each prototype and instance pair in the child structure's path leading back to its root structure. The root structure is shown first, then each subsequent child prototype and instance in the path. If the field owner is not a child structure then the structure path is this row has a gray background. The column is not displayed if there is no child structure table field owner in any row of the editor. For example, note the row in Figure 64 for the table "ahrs_M_BeaconedTimestamp.beaconedTimestamp". Since its **Structure Path** column is not empty, the table is a child structure. Working upwards from the bottom of the list, "beaconedTimestamp" is a child of the structure "ahrs_M_OutData" (a structure that is of prototype "ahrs_M_OutData_T"), which in turn is a child of the root structure "ahrs_OutDataPacket_T".

The remaining columns in the editor show the contents of the data fields chosen in the selection dialog. A cell with a gray background indicates that the associated table does not have the data field indicated by the cell's column; these cells may not be edited. A yellow background means that another cell or cells in the same column has an identical, non-blank value. The rows can be sorted by selecting the column headers, as with other table editors in the application. Column order can be changed by dragging a column to a new position.

Below the editor table are a number of filter check boxes that determine the type(s) of data fields displayed in the table. All of the filters are initially selected. The filters are as follows:

Project Display the project-level data fields (see paragraph 4.9.3.15).

Tables Display data fields belonging to the data tables (see paragraph 4.9.3.2.5.1).

Groups Display group data fields (see paragraph 4.9.3.9).

Table types Display the default data fields; those belonging to a table type definition (see paragraph 4.9.3.10.4.1).

The data fields to display can be changed by pressing the **Select** button, causing the initial data field selection dialog to reappear. However, if there is an unstored change or field marked for removal a confirmation dialog appears first, allowing the user to choose between continuing with the selection operation and discarding the changes, or canceling it and retaining the current selection with its unstored changes. The current selection state of the data field type filter check boxes is retained when the newly selected fields are displayed.

Data field values can be altered or the entire field removed via the editor. To change a field's value highlight the cell and press the Enter key, or double left-click the mouse while the pointer is over the cell. The data type constraints set when the field was created (e.g., hexadecimal or positive integer) are enforced for the new field values. To remove a field entirely select the field using the mouse and press the **Remove** button. The field's background is displayed in red to indicate it is marked for removal. Multiple fields can be selected for removal. Selecting a marked field and pressing **Remove** again unmarks the field for removal, and the background color returns to normal.

If one or more cells is selected and the **Open** button is pressed then the table(s) associated with the selected cell(s) are opened in a table editor. A row is ignored if it contains a data field belonging to a group or table type.

Selecting the **Print** button opens a printer selection dialog in order to print a copy of the editor table to the selected printer or file.

Unstored data field edits and removal selections can be undone by pressing the **Undo** button. Changes are undone in the order they were input. The Ctrl-Z key sequence performs the identical function. Conversely, undone changes can be reentered by pressing the **Redo** button or by the Ctrl-Y key sequence.

The **Store** button must be pressed to update the project's database with the data field value changes and removals. A confirmation dialog appears allowing the user to choose between continuing with the store operation and canceling it.

Selecting **Close** closes the data field editor dialog. If a change has been made to a data field that hasn't been stored in the project database, or there are one or more fields marked for removal then a confirmation dialog appears allowing the user to choose between continuing with the close operation and discarding the changes, or canceling it and keeping the editor open.

4.9.3.17 Padding

The **Padding** command causes a dialog to be displayed (Figure 65) that provides for adding or removing padding variables from the structure tables to properly byte align the structure variables. This command is enabled only for a user with read/write or administrator access. The dialog's table tree displays the prototype structure tables. One or more tables must be selected before adjusting the padding. Selecting the **Prototypes** node in the table tree selects all structure tables.

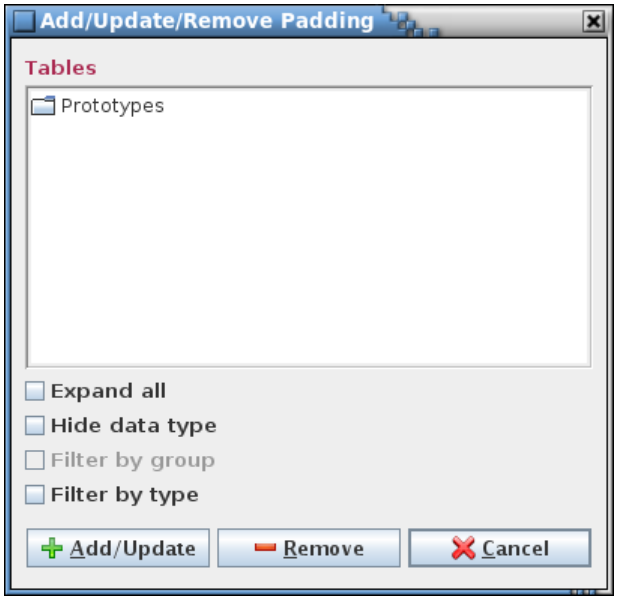


Figure 65. Padding adjustment dialog

The padding adjustment button commands are summarized below:

Add/Update Inserts padding variables into the selected prototype structure tables as needed to align the variables based on the size of the largest element within the structure (including any referenced child structures). The padding variables are of data type *char* and are represented as a single variable or an array of variables (if needed for multiple, consecutive padding). The exception for the data type is that padding variables added to "fill out" the unused bits for a bit-wise variable or series of bit-packed variables has the same data type as the bit-wise variable(s). The padding variable names are in the format *pad#__* (where *#* is one or more numerals). Padding variables are highlighted in the structure tables; see Figure 66. These variable rows may be manually edited as with any other row – added, altered, or removed – but are only recognized as the automatically inserted variety if they conform to the aforementioned name and data type constraints. If any data table has unstored changes a confirmation dialog appears allowing the user to choose between continuing with the padding add/update operation (losing the unstored changes) and canceling it (allowing the changes to be stored).

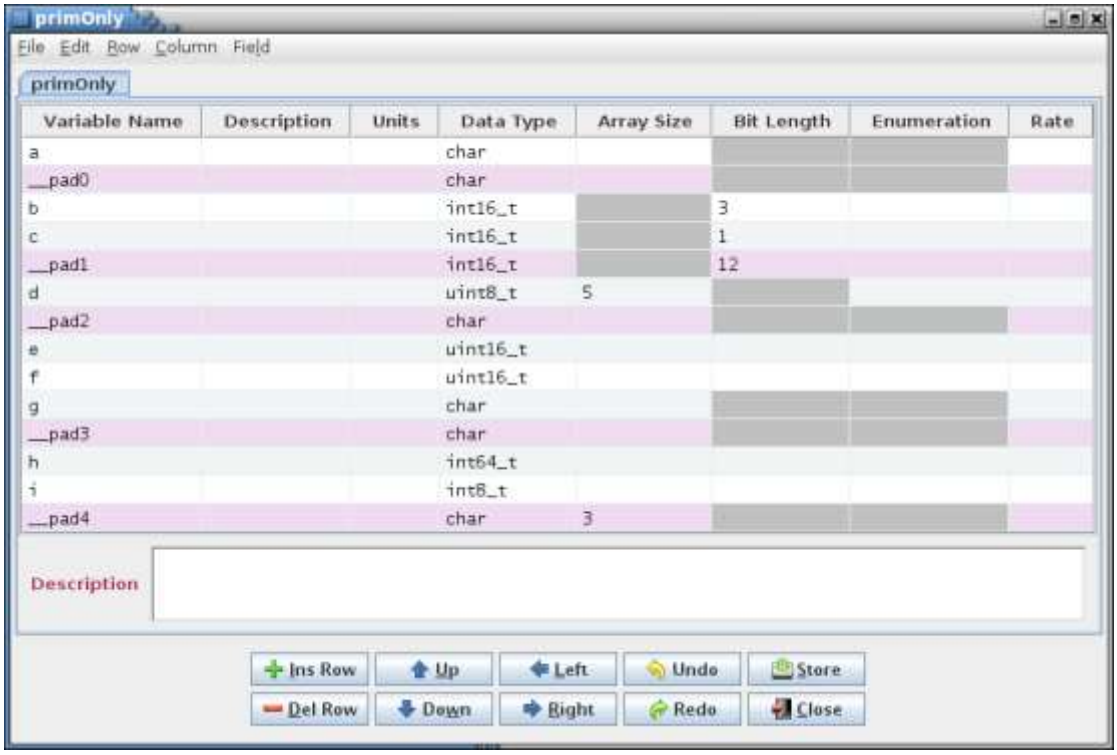


Figure 66. Structure table showing highlighted padding variables

Remove Causes all padding variables to be removed from the selected prototype structure tables. Only variables conforming to the name and data type constraints outlined in the description of the **Add/Update** button are recognized as padding variables. If any data table has unstored changes a confirmation dialog appears allowing the user to choose between continuing with the padding removal operation (losing the unstored changes) and canceling it (allowing the changes to be stored).

Close Closes the padding adjustment dialog.

While the padding is being added/updated or removed a dialog appears (Figure 67) showing the adjustment progress. This dialog allows halting padding adjustment by pressing the **Halt** button. Padding may be altered in the project database for some of the structures depending on when in the adjustment process the **Halt** button is pressed.

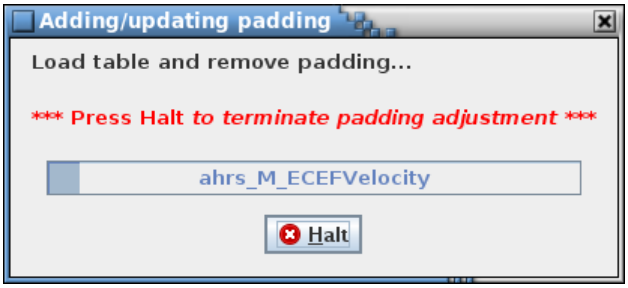


Figure 67. Example padding adjustment progress/cancellation dialog

4.9.3.18 Show variables

The **Show variables** command causes a dialog to appear that displays the variable paths and names, in alphabetical order, for tables representing structures in the project database (see Figure 68 for an example). The variables displayed can be constrained by selecting one or more tables in the table tree.

Only variables for the selected tables are displayed. Selection of a header node in the table tree (e.g., 'Parents & Children', or a group name if group filtering is enabled) selects all tables under that header. If no table or header node is selected in the tree then all variables in the project are displayed. The number of variables matching the constraints is displayed beside the variable table's label.

The paths and names are shown in two formats. The **Application Format** column shows the format used within the application:

rootTable
[,*structureDataType1*.*structureVariableName1*
[,*structureDataType2*.*structureVariableName2*
[,...]]],
primitiveDataType.*variableName*[*[arrayIndex]*]

The variable path and name is a combination of the structure's root table, ancestor structure(s) (if any), and the variable data type and name. This combination is unique for each variable defined in the project database. Note that any macro embedded in a variable name is replaced by its corresponding value before being displayed in the dialog.

The **User Format** column shows the path and name based on the user inputs. The **Enter variable path separator character(s)** text field allows entering the character(s) that are used to replace the commas that separate each variable in the variable path. The **Enter data type/variable name separator character(s)** text field allows entering the character(s) that are used to replace the periods that separate the data types and variable names. The **Hide data types** check box, if selected, causes the data types (structure and primitive) to be removed from the path, along with the periods that separate the data types from the variable names. The data type/variable name separator field is disabled and ignored for this case. Array member indices are altered by replacing the left bracket ([) with an underscore (_) and removing the right bracket (]). The **User Format** column values are identical to that returned by the script data access method call (see Table 10):

```
getFullVariableName("variable path + name", path_separator,
hide_data_types, type_and_name_separator)
```

To perform a conversion enter the separator character(s) and set the data type check box, then press the **Show** button. The **User Format** column updates to display the variables in the new format. The **Print** button outputs the table to a user-specified printer or file. The **Store** button stores the separators and show/hide data type state as a program preference so that these are restored if the dialog is closed and reopened. Additionally, this sets the separators used in displaying the variable path column; any open tables displaying a column with the variable path input data type are automatically updated. Select the **Close** button to exit the dialog.

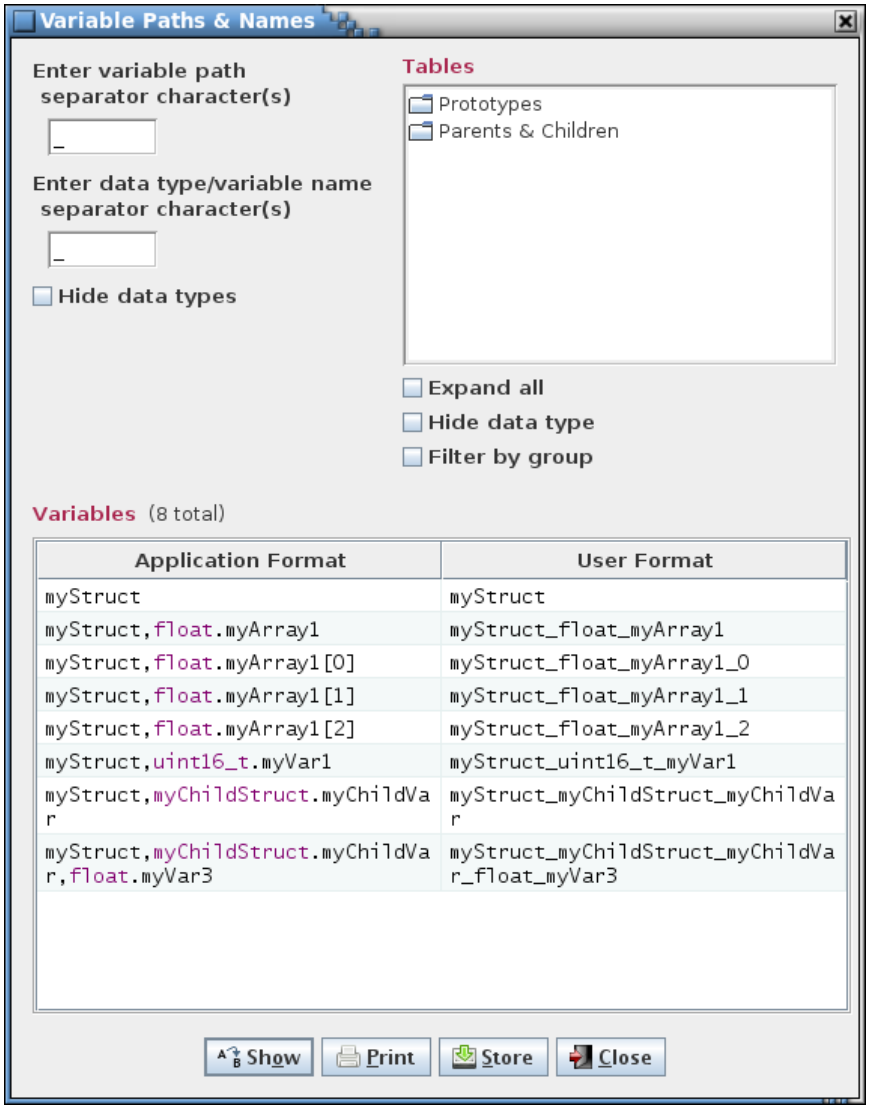


Figure 68. Example variable paths & names dialog

4.9.3.19 Show commands

The **Show commands** command causes a dialog to appear that displays information for commands defined in the project database (see Figure 69 for an example). The command information is displayed in alphabetical order, based on the command name (then on command code, and finally on command table name). The commands displayed can be constrained by selecting one or more tables in the table tree. Only commands for the selected tables are displayed. Selection of a header node in the table tree (e.g., 'Commands', or a group name if group filtering is enabled) selects all tables under that header. If no table or header node is selected in the tree then all commands in the project are displayed. The number of commands matching the constraints is displayed beside the command information table's label.

The **Command Name**, **Command Code**, and **Command Table** columns show the command's name, code, and table respectively. The **Arguments** column displaye the command's argument names, if any.

Initially all commands are displayed. Select or deselect tables in the table tree and press the **Show** button to update the displayed commands. The **Print** button outputs the table to a user-specified printer or file. Select the **Close** button to exit the dialog.

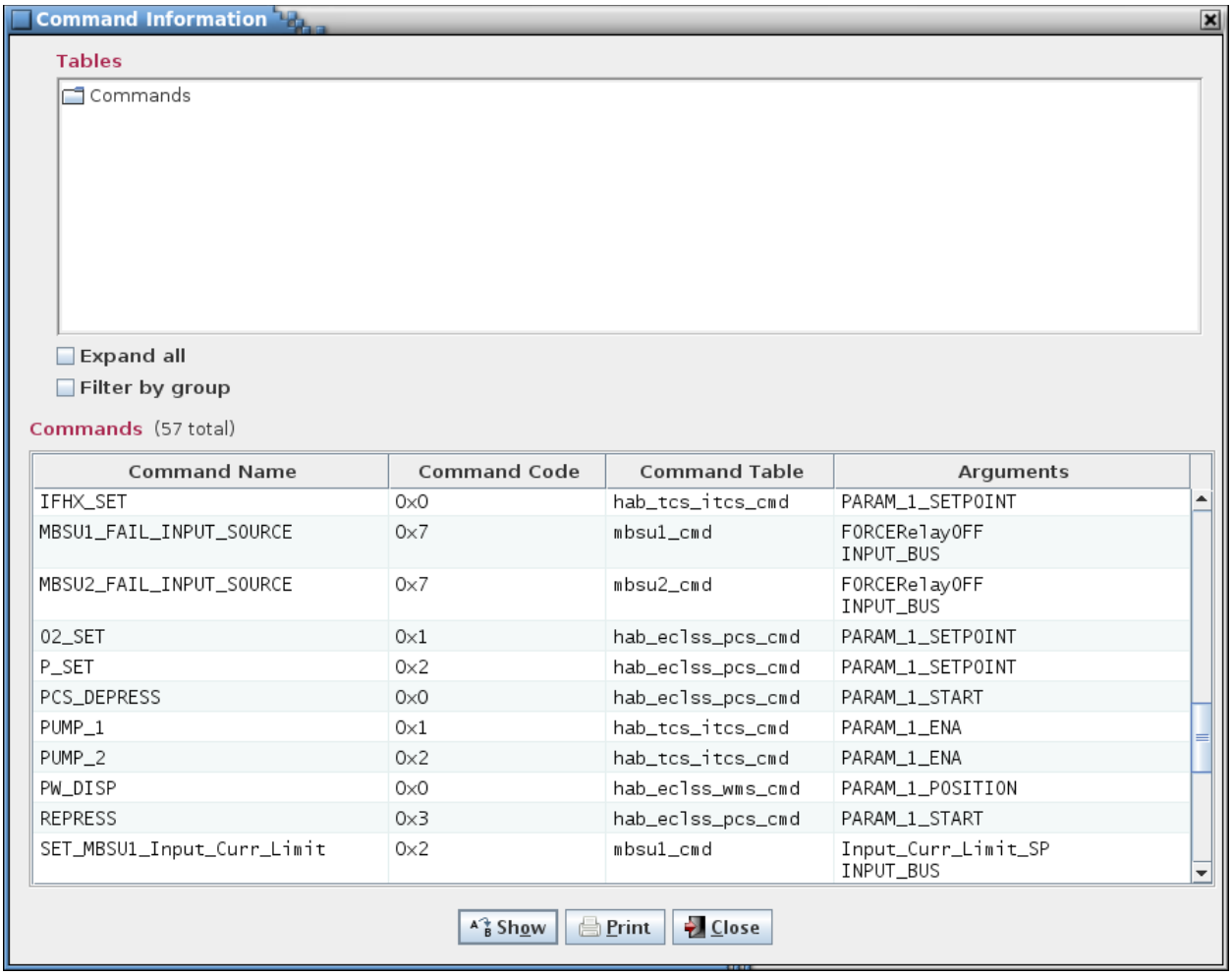


Figure 69. Example command information dialog

4.9.3.20 Search tables

The **Search tables** dialog provides a means of searching the project database data and internal tables for a specified text string (see Figure 70). Case sensitivity for the search is governed by the **Ignore text case** check box.

The **Allow regular expression** check box, when checked, allows the use of a regular expression to define the search pattern in the search text field. A regular expression can be formulated to find multiple matching conditions (for example, the search for **a.c** would match any string that has a single character between the characters 'a' and 'c'). Information regarding the use of regular expressions is beyond the scope of this document; however, resources and tutorials can be found online. When unchecked, the search text is matched as typed in the search text field.

The **Search data table cells only** check box, if selected, only displays matches found within the project database's data table cells and ignores those in the internal tables (see Appendix E.4; data table cell values stored in the custom values tables are included in the search).

The search can be constrained by selecting one or more tables in the table tree. Only matches in the selected tables are reported. Selection of a header node in the table tree (e.g., 'Parents & Children', or a group name if group filtering is enabled) selects all tables under that header. If no table or header node is selected in the tree then all tables are searched.

Enter the search text in the input field and select the **Search** button. The search results are displayed in the dialog's **Search results** table. The number of results is displayed beside the results table's label. The first column, **Owner**, shows the name of the data table or data object (table type definition, data field, group, script association, link, telemetry message, or scheduler entry) where a match is found. The second column, **Location**, describes the location of the match in the table/object. For a table the location is the column name in the table. A data object location depends on the type of object. For a data field this can be the field name, description, etc., whereas for a group or link this can be one of the tables or variables belonging to the group/link. The last column, **Context**, displays the string from the table or object containing the search text, with the search text highlighted.

The search text field uses auto-completion to fill in the search string. The previous search strings (those for the event log, table, and script) are remembered, including those from previous sessions. The number of remembered search strings can be changed via the Preferences dalog, and defaults to 30. Case sensitivity for auto-completion is based on the **Ignore text case** check box selection state.

The input text can be changed and the **Search** button pressed again to initiate another search of the tables. If one or more cells is selected and the **Open** button is pressed then the table(s) associated with the selected cells are opened in a table editor. A row is ignored if it contains a reference to other than a table. The search results can be output to a file or printer by selecting the **Print** button. To exit the search dialog select the **Close** button.

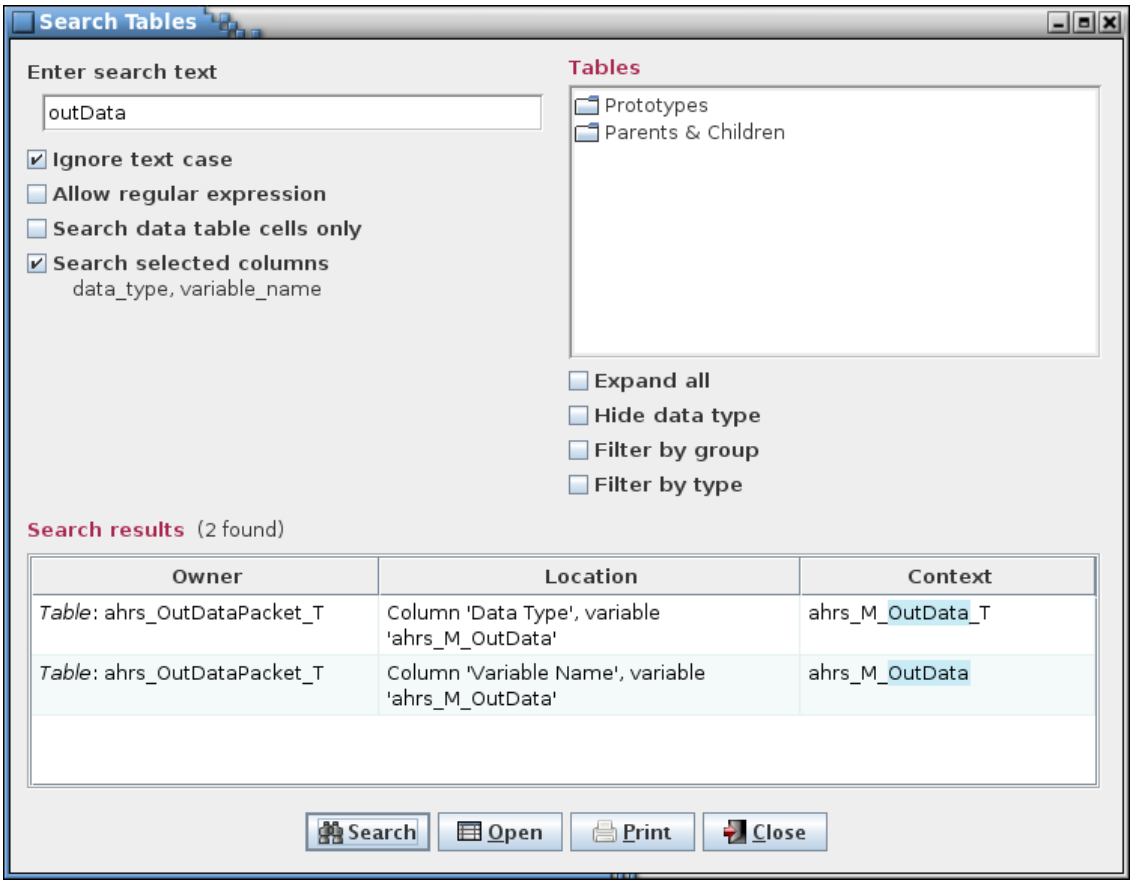


Figure 70. Search tables dialog

4.9.4 Scheduling

The scheduling commands are used to create and manage the information required to schedule telemetry downlink and application execution.

4.9.4.1 Manage links

The **Manage links** command opens the Manage Links dialog (Figure 71). This command is disabled if no rate columns are defined. The link manager allows the user to create telemetry parameter linkages. These are simply groupings, selected by the user, of telemetry parameters (i.e., variables in the structures) with the same sample rate. The link information is used when assigning variables to telemetry messages in the telemetry scheduler (paragraph 4.9.4.2) to force the linked variables to be contained within the same message(s). The linkages created are specific to the data stream to which the linkage belongs. In other words, variables that are linked in one data stream do not have to be linked in another data stream.

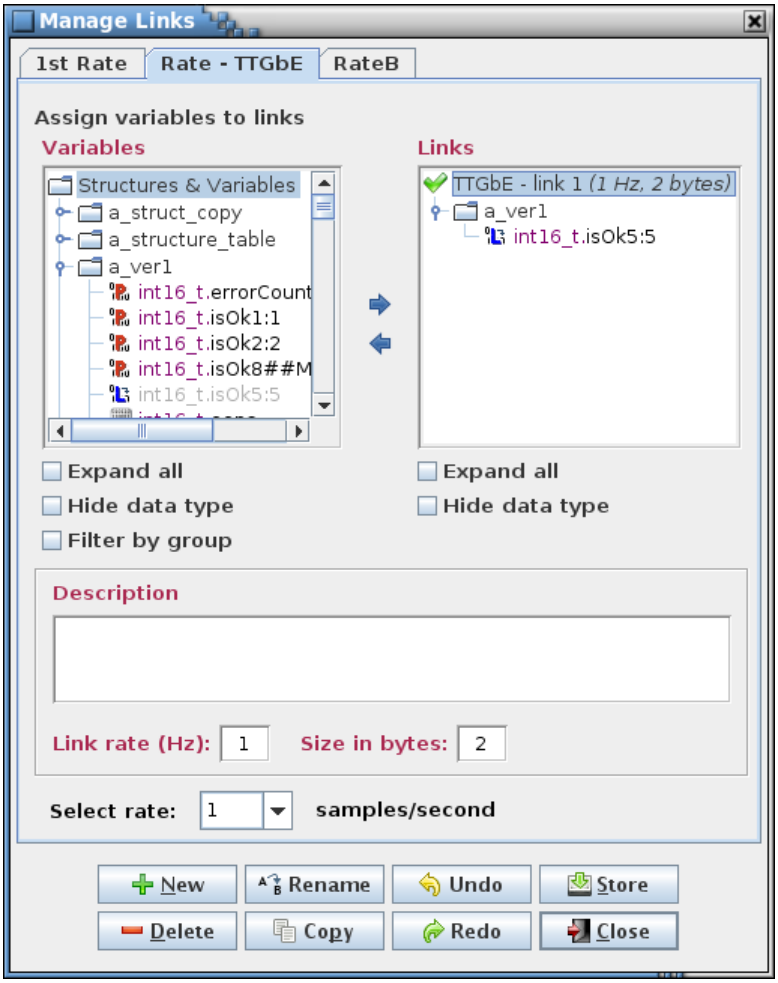


Figure 71. Manage Links dialog

The dialog's components are as follows. Along the top are the tabs that allow selection of the data stream in which to create, alter, or delete linkages. The upper left displays a tree showing structures and their members (under the heading **Variables**), both child structures and primitive variable types. The variables that are displayed in the tree are determined by the rate chosen from the **Select rate** combo box pull down menu near the bottom of the dialog. Rate values are grayed out and can't be selected if there is no variable that has that rate assigned. In the upper right is a tree showing the links and their member variables (under the heading **Links**). Between the trees are left and right arrows for adding or removing a variable from a link. Each tree also has one or more check boxes, to expand/collapse the tree and to filter the tree information. Below the trees is an input field for providing a description of a link. Underneath this is the link rate, in samples per second, and the total

size in bytes of the link, which is the sum of the byte sizes of the variables assigned to the link. The description, rate, and size fields are active when a single link is selected in the link tree.

The space separating the variable and link trees delineates a split pane control that is used to resize these panels relative to one another. Position the mouse pointer between the two tree panels and when the pointer changes to a double-headed arrow press and hold the left mouse button. Space permitting, the adjoining panes can be resized by moving the mouse pointer left or right. Release the mouse button to exit resizing.

In the link tree, displayed in parentheses next to each link name, are that link's rate and size in bytes (the same information that appears below the description field when this link is selected). A link's rate must match the selected sample rate (or the link must have no variables assigned) in order for it to be assigned variables from the variable tree. A check mark (✔) beside the link name indicates that the link can be assigned variables from the variable tree, and a red X (✘) is displayed if the link is incompatible with the selected sample rate (the tree text is also grayed out for incompatible links).

A variable may not be assigned to more than one link for a given data stream. Once assigned to a link the variable still appears in the variable tree but it is disabled (grayed out and not selectable). When a variable is removed from a link it becomes enabled again in the variable tree. Once an entire structure's complement of variables is assigned the structure itself it disabled in the variable tree, and if all structures are assigned then the **Structures & Variables** tree node itself is disabled.

If a variable is selected in the **Variables** tree then the link to which it belongs is selected automatically in the **Links** tree. Since linked variables are disabled in the **Variables** tree the variable isn't highlighted when selected. Selecting a non-linked variable deselects any highlighted link in the **Links** tree.

To create a link select the **New** button and provide a link name and, optionally, a description, in the input dialog that appears (Figure 72). The description can be altered later in the main dialog. The new link name appears in the link tree. The link name may not be blank, nor is the name allowed to match that of an existing link in the selected data stream. The link name may contain alphanumeric, space, and punctuation characters. There is no constraint on the length of the name.

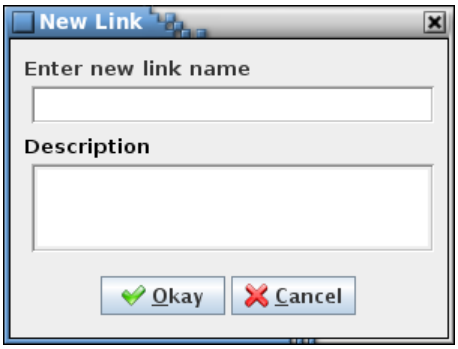


Figure 72. New Link dialog

To add variables to a link select the link in the link tree using the mouse or keyboard. Expand the variable tree as needed and select one or more variables using the mouse or keyboard. Multiple variables can be selected simultaneously by holding the Ctrl or Shift keys down when making a selection. Selecting a structure automatically includes its child structures (and their children, etc.), and all variables associated with the structure(s). Choosing a child structure automatically includes its parent structure, and its parent's parent, etc., up to its root structure, but does not include any of its sibling variables (i.e., a variable having the same parent structure and at the same tree level as the chosen variable). The exception is if the selected variable is bit-packed with one or more variables; in this case all of the packed variables are automatically included (see paragraph 4.5.5). Finally, select the right arrow button

in the center of the dialog. The variable(s) chosen appear in the selected link, and the link's tree is expanded to show the variable(s) added. Note that the variable hierarchy is preserved in the link's tree. More variables can be assigned to the link as described above.

To remove structures or variables from a link expand the link's tree and select the structure(s) and/or variable(s) to remove using the mouse or keyboard. Then select the left arrow button in the center of the dialog to delete the structure(s) or variable(s) from the link. A structure's children (and their children, etc.) and variables are removed along with the chosen structure. If a bit-packed variable is removed then all other variables packed together with it are removed as well, even if not explicitly selected.

To delete a link, first select it in the link tree, then select the **Delete** button. Multiple links can be removed simultaneously if desired by highlighting them while using the Shift or Ctrl keys.

To rename a link, select a single link from the link tree, then press the **Rename** button. An input dialog appears with the name of the selected link in the input field. Alter the name as desired and select **Okay** to change the link's name. The renamed link name may not be blank, nor is the name allowed to match that of an existing link in the current data stream. Select **Cancel** to exit the input dialog without affecting the link's name.

A link, including its description, variable structure(s) and variable(s), can be copied from one data stream to another. First select one or more links from the link tree to be copied, then press the **Copy** button. A dialog appears (Figure 73) with the name(s) of the selected link(s) in the link name text field. Below the link name field is an array of check boxes, one for each of the project's data stream names. The current data stream is grayed out and can't be selected (recall that a variable may belong to only one link in a given data stream). Select one or more data streams to which the link (or links) is to be copied. Press the **Copy** button to copy the link(s) to the selected data stream(s). Select **Cancel** to exit the copy dialog without copying the link.

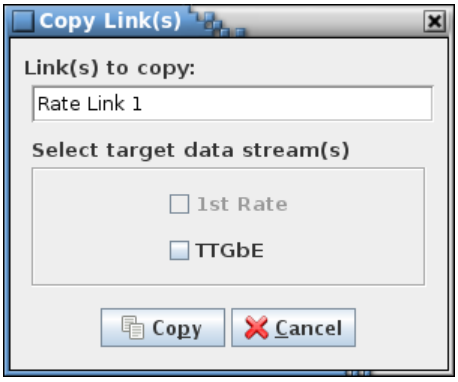


Figure 73. Copy Link(s) dialog

If the targeted data stream doesn't support the link's sample rate or the link name already exists in the stream then the link isn't copied to that stream. If a variable's sample rate differs between the copied stream and the target stream, or if the variable is unavailable in the target stream (i.e., the structure containing the variable doesn't have the rate column corresponding to the target data stream) then the variable isn't copied. For these cases a dialog is displayed indicating which link(s) or link member variable(s) could not be copied to which stream(s) and the reason for the failure(s) (see example in Figure 74). The **Print** button allows outputting the coy failure table to the selected printer or file. The **Close** button exits the copy failure dialog.

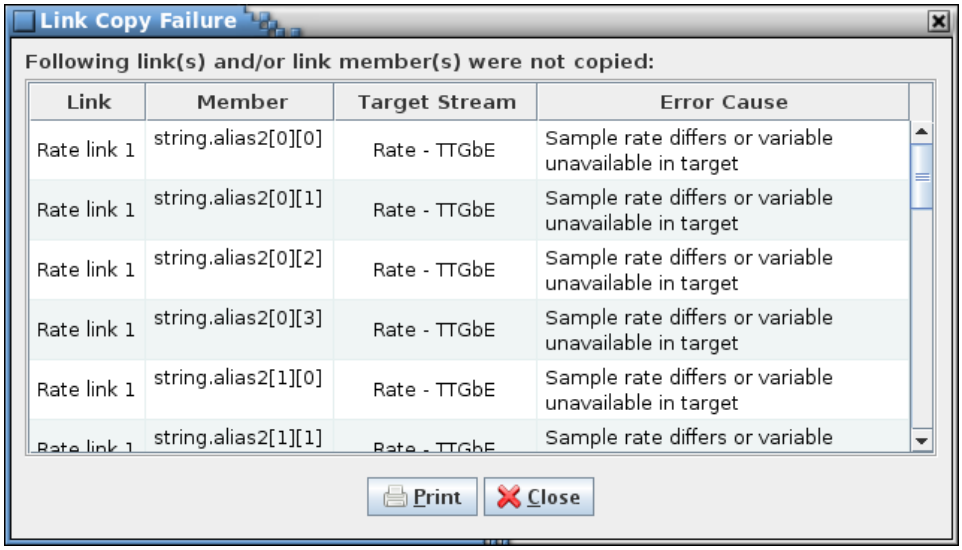


Figure 74. Example link copy failure dialog

A link's description can be added or changed by first selecting the link in the link tree. The current description for the link appears in the **Description** input field. The description can then be changed as desired.

Changes to the links (descriptions and member variables) for all data streams are stored in the project database only when the **Store** button is pressed. If changes have been made a confirmation dialog first appears. Select **Okay** to store the updates; select **Cancel** to exit the confirmation dialog without altering the project database.

Select the **Close** button to exit the link manager dialog. If there are any unsaved link changes in any of the data streams a dialog appears requesting confirmation to discard the changes. Select **Okay** to exit the link manager, losing any unsaved changes. Select **Cancel** to return to the link manager dialog.

The link manager button commands are summarized below:

New Create a new link.

Delete Delete the selected link(s).

Rename Rename the selected link.

Copy Copy the selected link, including its member tables, so another data stream.

Undo Undoes the last action performed (table assignment, typing, paste, insert, delete, redo, etc.) on the selected link.

Redo Reverses the last action undone (table assignment, typing, paste, insert, delete, undo, etc.) on the selected link.

Store Stores the changes made to the links in the link manager into the project database.

Close Closes the link manager window. If any changes have not been stored then a dialog appears allowing the user to confirm discarding the updates or to cancel closing the editor.

4.9.4.2 Telemetry

The **Telemetry** command opens the Telemetry Scheduler dialog (Figure 75). This command is disabled if no rate columns are defined. The telemetry scheduler is used to assign a project's variables to telemetry messages. The message information can be used to build a CFS housekeeping "copy table" (for

example, by using the copy table script provided with the CCDD application). The available messages are determined by the rate parameters. These parameters can be altered in the Rate Parameters dialog (see paragraph 4.9.4.4). Before the telemetry scheduler can be used the following must be done:

- Adjust the rate parameters to establish the correct boundaries for handling the project's telemetry
- Assign rates to the variables to be downlinked in the Edit Table dialog (see paragraph 4.9.3.2)
- (*Optional*) Assign variables that are desired to be sent down in the same message to a link using the link manager (see paragraph 4.9.4.1)

Figure 75. Telemetry Scheduler dialog

The Telemetry Scheduler dialog is composed of number of components. Along the top are tabs for each defined data stream (see paragraph 4.8). Each stream has its own variable rates and message assignments. Selecting a tab displays the dialog components associated with that data stream. At the top left is displayed the total number of bytes remaining to be assigned. This value is equal to the maximum bytes per second (from the Rate Parameters dialog) minus the size in bytes of all the variables assigned to messages. At the upper right is the cycle time which is the amount of time it will take for the messages to repeat; e.g., a cycle time of 2 seconds means that each message in the table will be sent down at least once every 2 seconds.

The **Variables** tree, at the left of the dialog, displays in tree format the variables available for downlink. Only variables assigned a rate are displayed in the variable tree. The tree is separated into two sections: **Linked Variables** and **Unlinked Variables**. **Linked Variables** displays the links per the currently selected data stream and rate filter. Each link contains the variables assigned to the link via the link manager. **Unlinked Variables** displays all the variables with a rate matching the selected rate filter. Variables assigned to a link are also displayed, but are grayed out and cannot be selected. Beneath the variable tree are two check boxes that are used to expand the tree or filter it by group.

To the right of the **Variables** tree is the **Options** list. This list displays the available options, based on the selected rate filter, for assignment of the variables to the messages. For example, if the rate filter is set to 5 and there are 10 available messages then the options displayed are "Option 1: Messages 1, 3, 5, 7, 9" and "Option 2: Messages 2, 4, 6, 8, 10" (assuming the cycle time is one second).

The **Rate Filter**, just below the **Options** list, is a pull down list of the data stream's available rates. The selected rate is used to filter the **Variables** tree and the **Options** list. Rate values are grayed out and

can't be selected if there is no variable in the data stream that has that rate assigned. The variable tree only displays variables that have a rate that matches the selected rate filter value. The user can change the rate filter at any time to make the **Variables** tree and **Options** list update.

The **Scheduler** table, located on the right of the **Options** list, contains a table with every available message. The scheduler table has at least three columns: the **Message** column, which displays the message name; the **Bytes** column, which displays the remaining bytes for each message; the **ID** column, which display the message's ID value. Extra columns, labeled **Sub 1**, **Sub 2**, etc. are added if any message has a sub-message; for messages without the specified sub-message the column is grayed out. The **Bytes** column is updated as variables are added or removed from the message. A negative number indicates that the message is over assigned (i.e., insufficient bytes available to contain the assigned variables); the **Message** column is displayed in red in this case. The message names and the ID (and sub ID) values can be edited in the **Scheduler** table, or can be automatically assigned in the Assign Telemetry Messages dialog called via the **Assign Msgs** button.

The **Assigned Variables** tree, located to the right of the **Scheduler** table, shows the variables assigned to the most recently selected message in the **Scheduler** table.

In between the **Options** list and **Scheduler** table are two arrow buttons. The right arrow button assigns one or more variables to a message. The assignment process is described below. The left arrow button **r**emoves one or more variables from a message. The removal process is described below.

The **Variables**, **Options**, **Scheduler**, and **Assigned Variables** portions of the telemetry scheduler can be resized. Position the mouse pointer between adjoining panes and when the pointer changes to a double-headed arrow press and hold the left mouse button. Space permitting, the adjoining panes can be resized by moving the mouse pointer left or right. Release the mouse button to exit resizing.

At the bottom of the telemetry scheduler dialog is the button panel. The button functions are as follows:

Auto-fill Assigns all the variables in the variable tree that are not yet assigned to messages. Auto-fill does this optimally so each message is filled as evenly as possible. During the auto-fill operation a progress/cancellation dialog appears (Figure 76). Pressing the dialog's **Halt** button stops the auto-fill operation; however, any variable assignments made up to that point are retained.

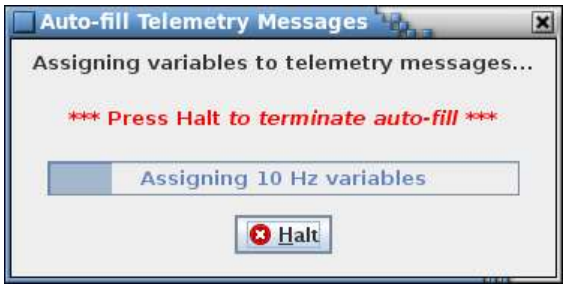


Figure 76. Telemetry message auto-fill progress/cancellation dialog

If auto-fill is successful then all the variables are assigned to an appropriate message. If auto-fill is unable to assign every variable (due to insufficient room or no available option) it displays a dialog indicating how many variables are left unassigned.

Assign Msgs Opens the Assign Telemetry Messages dialog (Figure 77). This dialog provides a means for assigning message names and/or message IDs to all messages and sub-messages based on a pattern, starting value, and interval value.

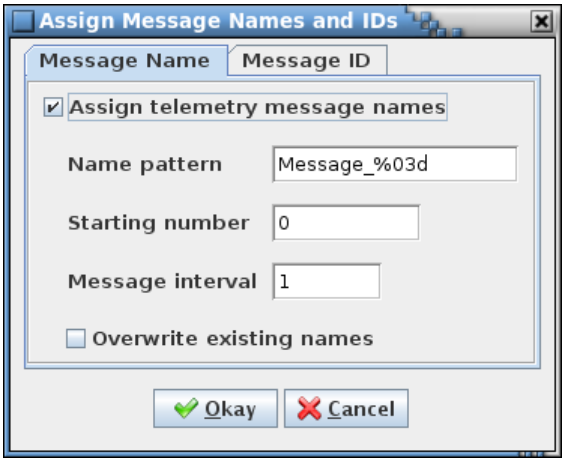


Figure 77. Assign telemetry message names and IDs dialog

To assign message names the **Assign telemetry message names** check box under the **Message name** tab must be selected. A pattern for the names is entered in the **Name pattern** input field. This pattern must adhere to alphanumeric naming constraints (see paragraph 4.7) except that it also must contain a single '%<0#>d' format string somewhere after the first character. The format string is replaced with a sequence number when the names are assigned. The optional '0#', where '#' represents one or more digits, provides a means of padding the sequence number with leading zeroes so as to bring its length to # digits. The first message name uses the pattern and the **Starting number** field value; the **Message interval** value is added to the previous message's number for each subsequent message name. For example, with the values as shown in Figure 77 the message names are "Message_001", "Message_002", "Message_003", etc., until all messages are named.

To assign message IDs the **Assign telemetry message IDs** check box under the **Message ID** tab must be selected. The **Starting ID** field is the starting ID number, in hexadecimal. The **ID interval** field is the interval used to calculate the next ID value in the sequence - the default is 1; any positive integer value is valid. The **Overwrite existing IDs** check box determines whether or not messages with an existing ID number are updated or left as is. The IDs are assigned beginning with the starting ID number and with each subsequent ID number equal to the previous number plus the interval value. Message IDs in the reserved message ID list (see 4.9.3.14.2) and IDs already assigned to message ID data fields for structure and command tables are automatically skipped when assigning IDs to the telemetry messages.

When **Okay** is selected the message names and/or IDs, based on the check box states, are assigned to the messages in the **Options** list and **Scheduler** table. Press **Cancel** to exit the dialog without changing the message names or IDs. Note that he telemetry scheduler's **Store** button must be used to update the messages in the project database. The names and/or IDs are assigned to the current data stream; the process must be repeated for each data stream. Avoid using the identical message ID name pattern for different streams since this can lead to duplicate output message ID names when creating the housekeeping copy table.

Clear Rate Removes all message and sub-message variable assignments for variables with a rate matching the currently selected rate filter.

Clear Msgs Removes all message and sub-message variable assignments.

Add Sub-msg Adds a sub-message to the message currently selected in the **Scheduler** table. Any number of sub-messages may be added. Adding a sub-message removes all of a message's sub-message variable assignments. This is done since the number of sub-messages affects the rate at which a sub-message is sent.

Del Sub-msg Removes the currently selected sub-message in the **Scheduler** table. Deleting a sub-message removes all sub-message variable assignments for that message. This is done since the number of sub-messages affects the rate at which a sub-message is sent.

Store Stores the telemetry scheduler data in the project database. Any changes not stored before closing the telemetry scheduler dialog are lost.

Close Closes the telemetry scheduler dialog. If there are any unsaved changes in any of the data streams a dialog appears requesting confirmation to discard the changes. Select **Okay** to exit the telemetry scheduler, losing any unsaved changes. Select **Cancel** to return to the telemetry scheduler dialog.

If a variable is selected in the **Variables** tree then every message to which it belongs is selected automatically in the **Scheduler** table, including sub-messages. The first message to which the variable belongs sets the tab in the **Assigned Variables** tree. The option corresponding to the message selection is highlighted in the **Options** list. Since assigned variables are disabled in the **Variables** tree the variable isn't highlighted when selected. Choosing a non-assigned variable in the **Variables** tree does not change the option and message selections.

The following describes the process to manually assign a variable to a message. First, one or more variables and/or links are selected in the **Variables** tree. A grayed out variable, structure, or link indicates that it is already assigned and can't be selected (linked variables also appear in the unlinked portion of the variable tree, but are grayed out and can't be assigned individually). After selecting one or more variables an option is chosen from the **Options** list. To aid in deciding which option to choose, the **Scheduler** table temporarily updates the **Bytes** column for the option's message(s), displaying the message size if that option is chosen. Also, the text of the **Message** column changes to either green, signifying there is enough room in the message for the variable(s), or red, signifying there is insufficient room. Changing which option is selected resets any of the temporary changes and updates the message(s) based on the new option. This allows the user to evaluate each option before selecting a choice. After deciding on an option, pressing the dialog's right arrow button assigns the selected variable(s) to each message in the selected option. Adding a linked or bit-packed variable also adds the variables associated with it; i.e., all members of the link are added, and all other variables bit-packed with the selected variable are added. Once a variable is assigned to a message it is grayed out in the **Variables** tree so it can't be assigned more than once.

A variable or variables can be removed manually from the messages and sub-messages. First a message or sub-message is selected (either in the **Scheduler** table or the **Assigned Variables** tree), which causes the **Assigned Variables** tree to display the variables for the selected (sub-)message. The user selects from the tree one or more of the variables or structures that are to be removed and then presses the dialog's left arrow button. The selected variable(s) is removed from the message(s) to which it is assigned. Removing a linked or bit-packed variable also removes the variables associated with it; i.e., all members of the link are removed, and all other variables bit-packed with the selected variable are removed. Any de-assigned variable is no longer grayed out in the **Variables** tree to signify it is available to be re-assigned.

4.9.4.3 Applications

The **Applications** command displays the Application Scheduler dialog (Figure 78). The application scheduler is used to schedule the execution frequency and order of a project's applications. The data

created from the application scheduler is used to create scheduler tables for the project. The schedule table is used by the CFS scheduler application (SCH) to determine when to execute the project's applications (demonstration scripts are provided that create the scheduler tables). The available time slots for when an application can be executed are determined by the application parameters that can be altered in the application parameters dialog (see paragraph 4.9.4.5). Before the application scheduler can be used the following must be done:

- Set the application parameters to establish the correct boundaries
- Create applications using the **Group Manager** dialog (see paragraph 4.9.3.9)

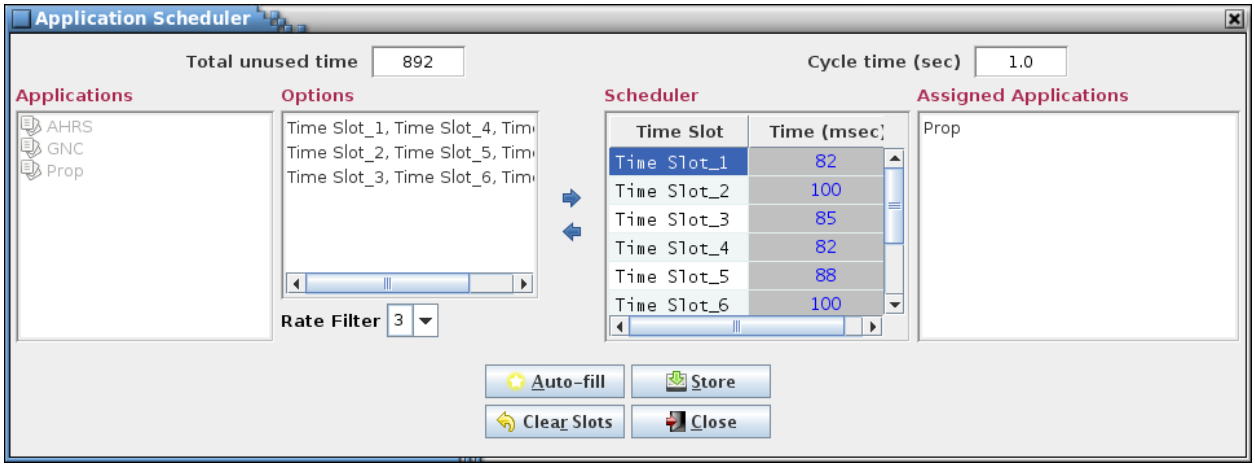


Figure 78. Application Scheduler dialog

The application scheduler dialog is composed of multiple components. At the top left is displayed the total number of milliseconds remaining to be assigned. At the upper right is the cycle time which is amount of time it will take for the schedule to repeat; e.g., a cycle time of 2 seconds means that the schedule table will be executed once every 2 seconds.

The **Applications** tree, at the left of the dialog, displays all the available applications to be scheduled. Any application that has already been assigned or has an execution rate that does not match the rate filter is grayed out and cannot be selected.

To the right of the **Applications** tree is the **Options** list. This list displays the available options, based on the selected rate filter, for assignment of the applications to the time slots. For example, if the rate filter is set to 1 and forty time slots are available then the options will be "Option1: TimeSlot_1", "Option2: TimeSlot_2", "Option3: TimeSlot_3", etc. for all forty time slots (assuming the cycle time is one second).

The **Rate Filter**, which is located below the **Options** list, contains a drop-down list of all the available execution rates. Changing the rate causes the **Applications** tree to gray out any applications that are not at the selected rate, and the **Options** list changes to display options for the newly selected rate.

The **Scheduler** table, which is located to the right of the **Options** list, is a table of all available time slots. The **Time Slot** column displays the time slots and the **Time (msec)** column displays the remaining available time for that time slot (in milliseconds). The available time decreases as applications are added to that time slot and increase if an application is removed. If the available time ever becomes negative (i.e., the sum of the assigned applications' execution times exceeds the maximum available for a time slot) then the time slot's text changes to red.

The **Assigned Applications** list, located to the right of the **Scheduler** table, displays the application(s) assigned to the currently selected time slot. If multiple time slots are selected only the first selected

time slot's applications are displayed. This allows the user a quick way to view the applications currently assigned to a time slot.

In between the **Options** list and **Scheduler** table are two arrow buttons. The right arrow button is used to assign one or more applications to a time slot. The left arrow button **r**emoves one or more applications from a time slot. The assignment and removal processes are described below.

The **Applications**, **Options**, **Scheduler**, and **Assigned Applications** portions of the application scheduler can be resized. Position the mouse pointer between adjoining panes and when the pointer changes to a double-headed arrow press and hold the left mouse button. Space permitting, the adjoining panes can be resized by moving the mouse pointer left or right. Release the mouse button to exit resizing.

At the bottom of the application scheduler dialog is the button panel. The button functions are as follows:

Auto-fill Assigns all the applications in the application tree that are not yet assigned to time slots. Auto-fill does this optimally so each slot is filled as evenly as possible. During the auto-fill operation a progress/cancellation dialog appears (Figure 79). Pressing the dialog's **Halt** button stops the auto-fill operation; however, any application assignments made up to that point are retained.

Figure 79. Application time slot auto-fill progress/cancellation dialog

If auto-fill is successful then all the applications are assigned to a time slot. If auto-fill is unable to assign every application (due to insufficient room or no available option) it displays a dialog indicating how many applications are left unassigned.

Clear Slots Removes all application time slot assignments.

Store Stores the application scheduler data in the project database. Any changes not stored before closing the application scheduler dialog are lost.

Close Closes the application scheduler dialog. If there are any unsaved changes a dialog appears requesting confirmation to discard the changes. Select **Okay** to exit the application scheduler, losing any unsaved changes. Select **Cancel** to return to the application scheduler dialog.

The following describes the process to manually assign an application to a time slot. First, one or more applications are selected in the **Applications** tree. A grayed-out application indicates that it is already assigned or doesn't have the same rate as that shown in the **Rate Filter**, and can't be selected. After selecting one or more applications an option is chosen from the **Options** list. To aid in deciding which option to choose, the **Scheduler** table temporarily subtracts the application run time(s) from the **Time (msec)** column for the selected option's time slot(s), displaying the time remaining if that option is chosen. Also, the text of the **Time Slot** column changes to either green, signifying there is enough room in the slot for the application(s), or red, signifying there is insufficient room. Changing which option is selected resets any of the temporary changes and updates the time slot(s) based on the new option. This allows evaluation each option before committing to a choice. After deciding on an option, pressing

the dialog's right arrow button assigns the selected applications(s) to each time slot in the selected option. Once an application is assigned to a time slot it is grayed out in the **Applications** tree so it can't be assigned more than once.

An application or applications can be removed manually from the time slot(s). First a time slot is selected in the **Scheduler** table; this causes the **Assigned Applications** list to display the applications for the selected slot. Select from the list one or more of the applications that are to be removed and then press the dialog's left arrow button. The selected application(s) is removed from the slots to which it is assigned. Any de-assigned application is no longer grayed out in the **Applications** tree, signifying it is available to be re-assigned.

4.9.4.4 Rate parameters

The **Rate parameters** command displays the dialog shown in Figure 80. This command is disabled if no rate columns are defined.

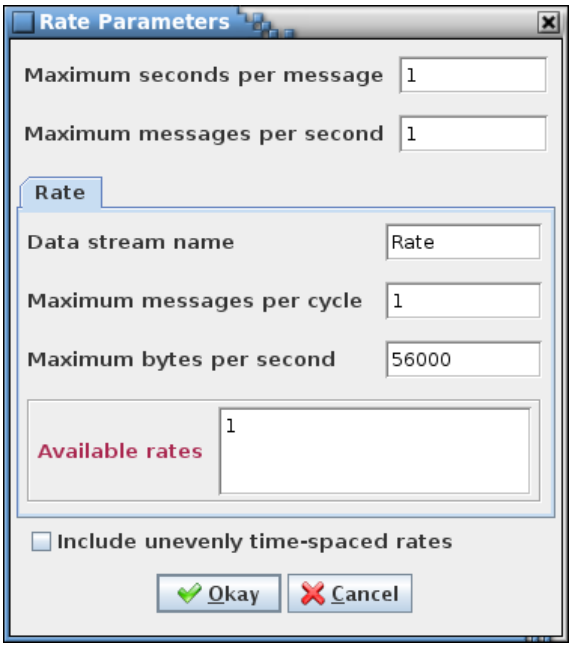


Figure 80. Rate Parameters dialog

This dialog is used to set the bounds for the sample rates for each defined data stream (see paragraph 4.8) and from these generate the selections in the drop down menu for the rate column(s) in the data tables, the link manager (see paragraph 4.9.4.1), and the telemetry scheduler (see paragraph 4.9.4.2). These parameters also define the total number of messages and maximum message size. Each of these parameters must be a positive, non-zero integer value. The first two parameters, maximum seconds per message and maximum messages per second, are common to all data streams, while the remaining parameters are assigned by data stream. A data stream is selected via the tabs, which reflect the rate column names, in the center of the dialog. The definitions of these values are as follows:

Maximum seconds per message The slowest period, in seconds, that a message is downlinked. Example: If 5 is entered then 5 seconds per sample is the slowest rate allowed to be selected as the rate for a telemetered value. All rates between this and 1 second/sample that are multiples of the period are added to the rate list. Rates slower than 1 sample per second are displayed in the format "1/#" where # is the number of seconds between samples.

Maximum messages per second Maximum number of telemetry messages that can be downlinked in a single second. For a cycle time of one second this value is the same as the **Maximum messages per cycle** value.

Data stream name This value is specific for a data stream. This is a user-defined alternate name to associate with the rate column and is used in the link manager and telemetry scheduler for the tab name (if no data stream name is entered the rate column name is used instead). The script access methods use the data stream name.

Maximum messages per cycle This value is specific for a data stream. This value is the number of telemetry messages that are downlinked during a single cycle through the message list. For a cycle time of one second this value is the same as the **Maximum messages per second** value.

Maximum bytes per second This value is specific for a data stream. This is the maximum number of bytes that can be downlinked during a single second.

The **Available rates** field displays the rates that are available based on the input values.

The evenly time-spaced sub-second rates are calculated using the above values. For example, given a maximum messages per cycle of 10 and a maximum messages per second of 10 then only rate values that are a factor of 10 – i.e., 1, 2, 5, and 10 samples per second – are available. The check box labeled **Include unevenly time-spaced rates**, when checked, causes the remaining, unevenly time-spaced rates to be included in the list of rates (in the example this is all values between 1 and 10 – i.e., 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 samples per second).

4.9.4.5 App parameters

The **App Parameters** command displays the Application Parameters dialog (Figure 81).

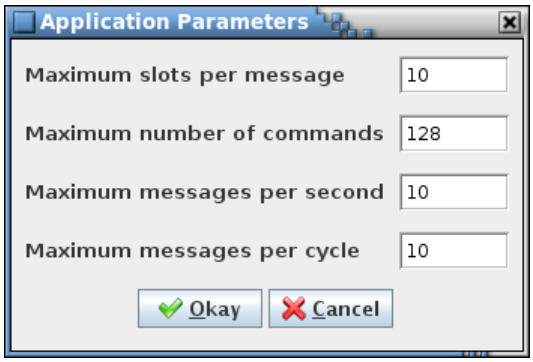


Figure 81. Application Parameters dialog

This dialog is used to set parameters for the application scheduler table. The values for **Maximum slots per message** and the **Maximum number of commands** define the boundaries, while the **Maximum messages per second** and the **Maximum messages per cycle** are used for scheduling the applications. Each parameter must be a positive, non-zero integer value. The definitions of these values are as follows:

Maximum slots per message The number of slots available in each time slot of the scheduler table**.** If 10 is entered then every time slot will have 10 available slots for an application. The Application Scheduler doesn't allow

	a time slot to have more applications assigned to it than this parameter.
Maximum number of commands	The maximum number of commands that can be created for the scheduler table.
Maximum messages per second	The maximum number of time slots that are available for an application to be scheduled in a second.
Maximum messages per cycle	The number of time slots that are executed during a single cycle through the time slot list. For a cycle time of one second this value is the same as the **Maximum messages per second** value.

4.9.5 Script

The **Script** menu contains commands for associating scripts with data tables and fields, and for executing the stored associations. Scripts are a means of accessing the project data in order to create output files (e.g., C header files or ITOS record files) or otherwise manipulate the data. The script languages supported by the application include JavaScript, Python, Ruby, Groovy, and Scala. Example scripts are provided with the application. These can be modified, or new scripts written as needed by the user. See paragraph 4.10 for more information on the use of scripts to access the table data.

The Manage Script Associations (see paragraph 4.9.5.1) and Execute Script(s) (see paragraph 4.9.5.2) dialogs are mutually exclusive; opening one causes the other, if displayed, to be closed.

4.9.5.1 Manage

The **Manage** command provides the means for associating scripts and data tables. This is required before executing the scripts. The associated scripts and tables can be stored in the project database so that frequently used associations can be quickly executed.

When the command is selected the Manage Script Associations dialog (Figure 82) is displayed. If the Execute Script(s) dialog (see paragraph 4.9.5.2) is currently displayed then it is closed. The dialog is divided into four sections: script selection, table selection, script associations, and command buttons. The line separating the table selection tree and script associations list delineates a split pane control that is used to resize these panels relative to one another. Position the mouse pointer over the separator line and when the pointer changes to a double-headed arrow press and hold the left mouse button. Space permitting, the adjoining panes can be resized by moving the mouse pointer up or down. Release the mouse button to exit resizing.

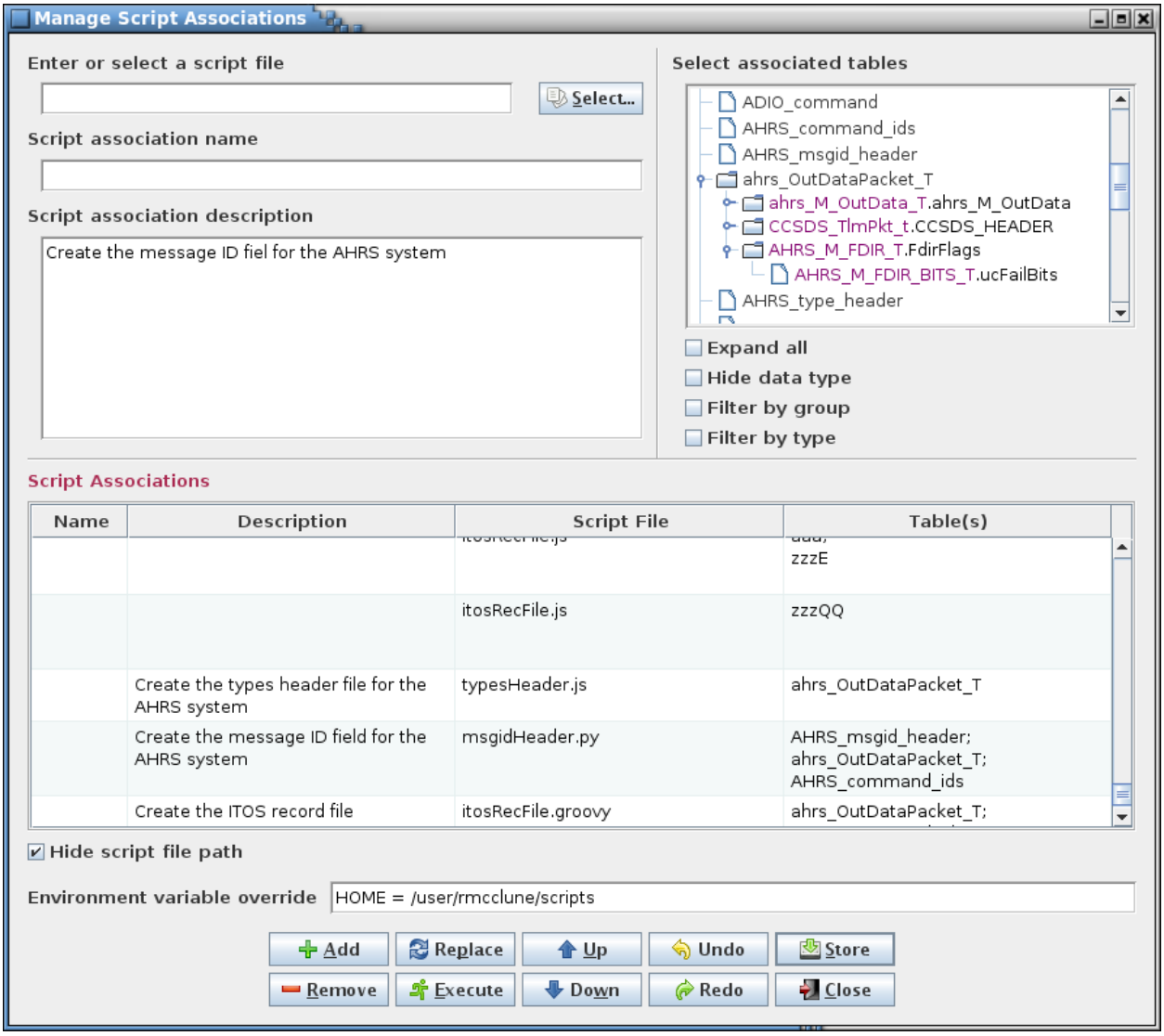


Figure 82. Manage Script Associations dialog

The script association name field allows a name to be assigned to an association. The name may not match one already in use by another association and must adhere to the alphanumeric input type (see paragraph 4.7). The association name is optional; it's purpose is to be used with the command line `execute` option (see paragraph 4.1) to reference an existing association, in place of typing the script name and table(s).

The script association description field allows a description to be added to an association. The description may remain blank.

The script selection field and **Select…** button are used to select a script file. A script name (with file path) can be typed into the field; alternatively, pressing the **Select…** button displays a file selection dialog from which a script file can be located and selected. Script names must be a valid for use as a file name (e.g., may contain spaces, but not certain special characters, dependent on the operating system, such as a forward slash (/)).

The table tree displays all of the root tables and their child tables (if applicable). The user expands the tree branches and selects one or more tables (see paragraph 4.5.3 for more information on table trees). When a structure table is chosen all of its child tables are automatically included when the script

association is executed; therefore the child tables do not have to be explicitly selected when creating the association, and any child nodes that are selected are eliminated from the **Script Associations** table's **Table(s)** column when the association is added. It makes no difference in what order the tables are selected for assignment since, when loaded for use by the script, the tables are sorted so that the root tables are in alphabetical order and the child tables appear in the order defined by their table type definition.

If the table tree is filtered by group then one or more group names can be selected from the tree for association with the chosen script. When the association is added to the table the group name (prepended with "Group:") appears in the **Table(s)** column. Any of the group's member tables that were also selected don't appear in the table, however. When the script association is executed the group's current member tables are loaded. This allows the script association to remain unchanged even if tables are added or removed from the group.

The **Script Associations** table displays the script associations that are stored in the project database, plus any that have been added while this dialog is open. Associations are grayed out if the script file doesn't exist on the local machine or an associated table doesn't exist in the project database. These disabled associations can be selected for removal, but can't be executed. The **Description** column may be edited to add, alter, or remove an association's description. The **Script File** column displays the association's script file and file path. Below the table is a check box that allows toggling between hiding and displaying the script file paths; this selection does not affect storing and retrieving the file paths and the check box's selection state is remembered between sessions. The **Table(s)** column display each table associated with the script in the **Script File** column. Each table is displayed on a single line (space permitting) and includes its full path.

Below the table is a check box, **Hide script file path**. When selected the file paths in the association table's **Script File** column are not displayed. Deselecting the check box restores the paths. The paths are used when executing the associations even if not displayed.

Script file paths are allowed to have environment variables within them. When the association is executed any variables in the script path are replaced by their corresponding value in the system environment variable map. The **Environment variable override** field allows the variable values in the system map to be temporarily replaced, or added if the variable doesn't exist in the map. The format for entries in this field is:

```
<key1 = value1<,key2 = value2<, …>>>
```

Each key/value pair must be separated by a comma. A dollar sign ($) can precede the key name, and spaces (except those embedded within a key or value) and double quotes bounding the keys and values are ignored. The contents of this field are stored as a program preference, so the field is automatically populated with the overrides that were last used.

Script associations may be executed from within this dialog. This is similar to execution of the associations from the Execute Scripts dialog (see paragraph 4.9.5.2) except that in this dialog the associations do not have to be stored in the project database to be executed. This provides a means to create a one-use association for immediate execution. See paragraph 4.9.5.2 for further detail on script execution.

The button commands are described below:

Add After entering a name (optional), description (optional), choosing a script file, and (if needed by the script) one or more data tables or groups, selecting the **Add** button creates the script association. The new association is inserted as the first row in the **Script Associations** table (any existing associations are moved down a row) and is automatically selected. A script may be used in more than one association, and a data

table or group may be used in any number of associations; however, duplicate associations, i.e., those utilizing the same script, table(s), and group(s), are not added to the list.

A script can be added without associating it with a table or group. This is the case when the script performs actions that do not need data from a specific data table (for an example see the script that creates the housekeeping (HK) copy table).

Remove An association may deleted from the table by selecting it using the mouse or keyboard, then pressing the **Remove** button. Multiple associations may be removed simultaneously by selecting more than one from the list by using the Ctrl or Shift keys.

Replace Replace the currently selected association in the **Script Associations** table with the association defined by the currently entered script name, description, script file, and selected table(s).

Execute Execute the script associations(s) that are selected in the script associations table (disabled associations are ignored). The application GUI is disabled during script execution in order to prevent possible alteration of the data while a script is accessing it.

Up Move the currently selected row(s) up one row relative to the remaining rows. The order of the script associations in the table affects the order in which the associations are executed if multiple associations are chosen to execute. Otherwise the capability to arrange the rows is solely for the user to group the associations as desired. The ordering is preserved when the associations are stored and retrieved from the project database, and can be useful for keeping affiliated associations near one another.

Down Move the row(s) of the currently selected cell(s) down one row relative to the remaining rows. The order of the script associations in the table affects the order in which the associations are executed if multiple associations are chosen to execute. Otherwise the capability to arrange the rows is solely for the user to group the associations as desired. The ordering is preserved when the associations are stored and retrieved from the project database, and can be useful for keeping affiliated associations near one another.

Undo Undoes the last action performed (add, remove, move up, move down, redo).

Redo Reverses the last action undone (add, remove, move up, move down, undo).

Store Stores the changes made to script associations in the script association manager into the project database. The order of the associations in the table is preserved. If changes have been made a confirmation dialog first appears. Select **Okay** to store the updates; select **Cancel** to exit the confirmation dialog without altering the database.

Close Closes the script association manager window. If there are any unsaved association changes a dialog appears requesting confirmation to discard the changes. Select **Okay** to exit script associations dialog, losing any unsaved changes. Select **Cancel** to return to the Manage Script Associations dialog.

4.9.5.2 Execute

Selecting the **Execute** command causes the Execute Script(s) dialog to appear (Figure 83). If the Manage Script Associations dialog (see paragraph 4.9.5.1) is currently open then it is closed unless there are unsaved changes; in this case a dialog appears requesting confirmation to discard the changes. If the user doesn't elect to discard the changes then the Manage Script Associations dialog remains open and the Execute Script(s) dialog doesn't appear.

The dialog displays a table of the stored script associations. Associations are grayed out if the script file script file can't be found in the folder indicated by the path or an associated data table doesn't exist in the project database. Grayed-out associations can't be selected for execution. Below the table is a check box that allows toggling between hiding and displaying the script file paths in the **Script File** column. See paragraph 4.9.5.1 for more information on the script associations table.

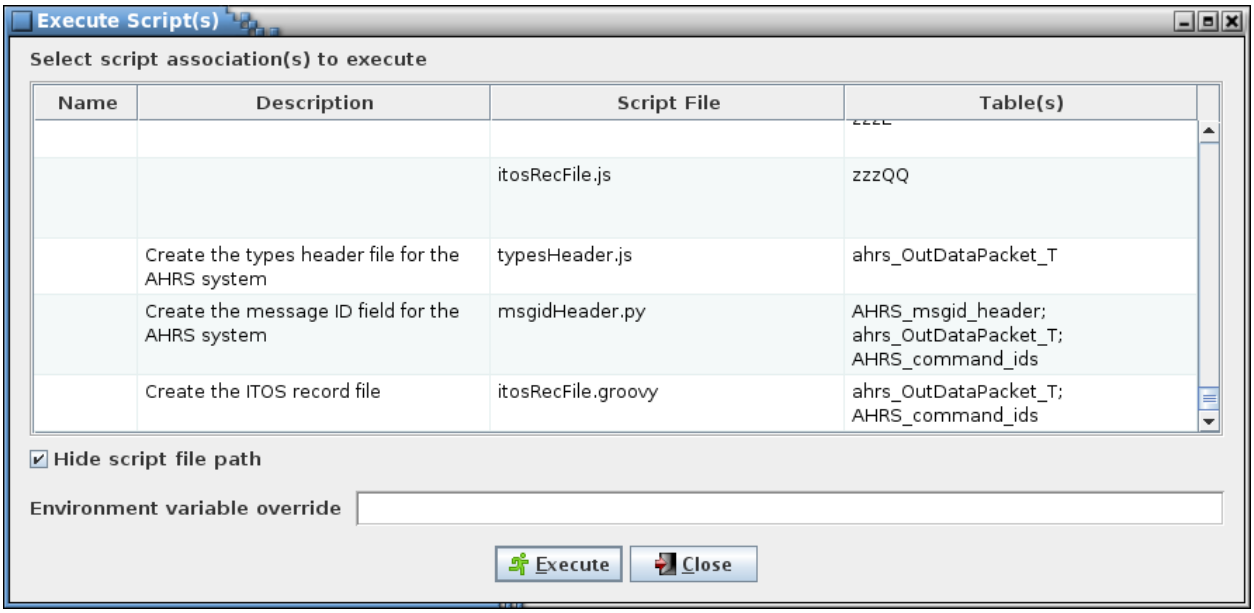


Figure 83. Execute Script(s) dialog

Select one or more associations from the table and press the **Execute** button to execute the selected association(s). The application GUI is disabled during script execution in order to prevent possible alteration of the data while a script is accessing it. While the script is executing the dialog shown in Figure 84 is displayed. Pressing the **Halt** button stops script execution immediately and returns control to the CCDD application – this is necessary, for instance, if the script contains an infinite loop.

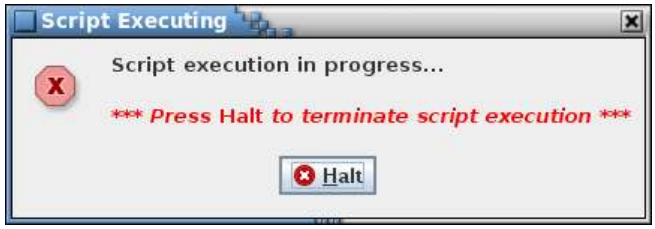


Figure 84. Halt script execution dialog

When script execution completes a status message is written to the event log. If script execution is halted the log entry indicates that the scripts failed to execute; however, if multiple scripts are executed it's possible that one or more successfully completed prior to the action to halt execution. If an error occurs, preventing successful script completion, an error dialog appears indicating that the script(s) failed to execute. The log entry in this case displays the script name(s) and provides details on the cause of the error.

Select the **Close** button to exit the Execute Script(s) dialog.

4.9.5.3 Store

The **Store** command is used to store scripts in the project database. This provides a means of script security and configuration management as well as allow all users, including those at remote sites, to

access a common set of script files. This command is enabled only for a user with read/write or administrator access. Selecting the command causes a file selection dialog to appear (Figure 85).

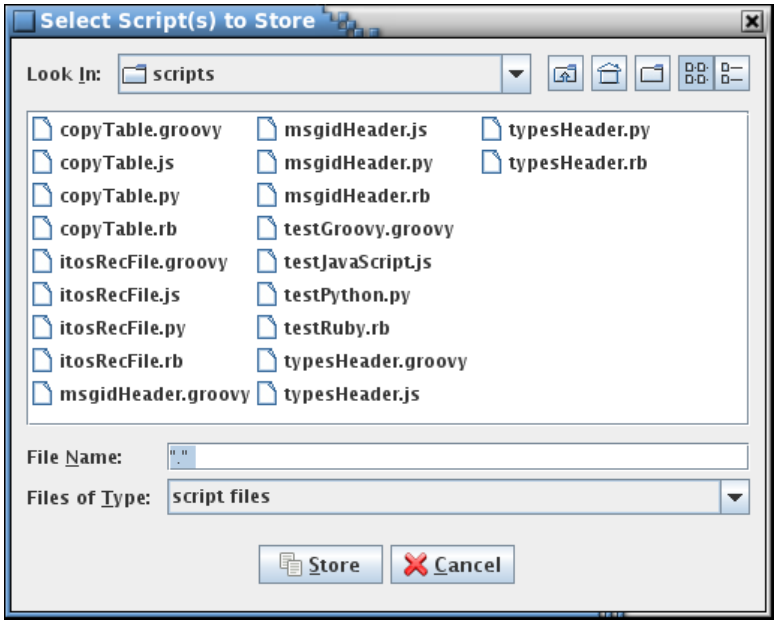


Figure 85. Script selection dialog

Only files with extensions supported by the available script engines are shown. However, other files are displayed if "All Files" is selected from the **Files of Type** drop down menu. After selecting one or more files, selecting the **Store** button stores the contents of the files in the project, each as a separate database table. Select the **Cancel** button to exit the dialog without storing any files. The **Retrieve** command (see paragraph 4.9.5.4) provides the means for retrieving the stored scripts from the project.

When a file is stored the application first searches it for the first line containing the text "description:". The search ignores case, so any combination of upper and lower case characters constitutes a match. If found, the remaining text on the same line in the file (sans any leading or trailing white space character(s)) is stored with the file as its description. The description appears alongside the file name in the Retrieve Script(s) dialog (Figure 86). If no match is found then the description text in the dialog is blank.

Note that this command can be used to store any text file in the project database, not only script files.

4.9.5.4 Retrieve

Selecting the **Retrieve** command causes the Retrieve Script(s) dialog to appear (Figure 86). This command allows for extracting scripts (or other text files) that are stored in the project database using the **Store** command (see paragraph 4.9.5.3).

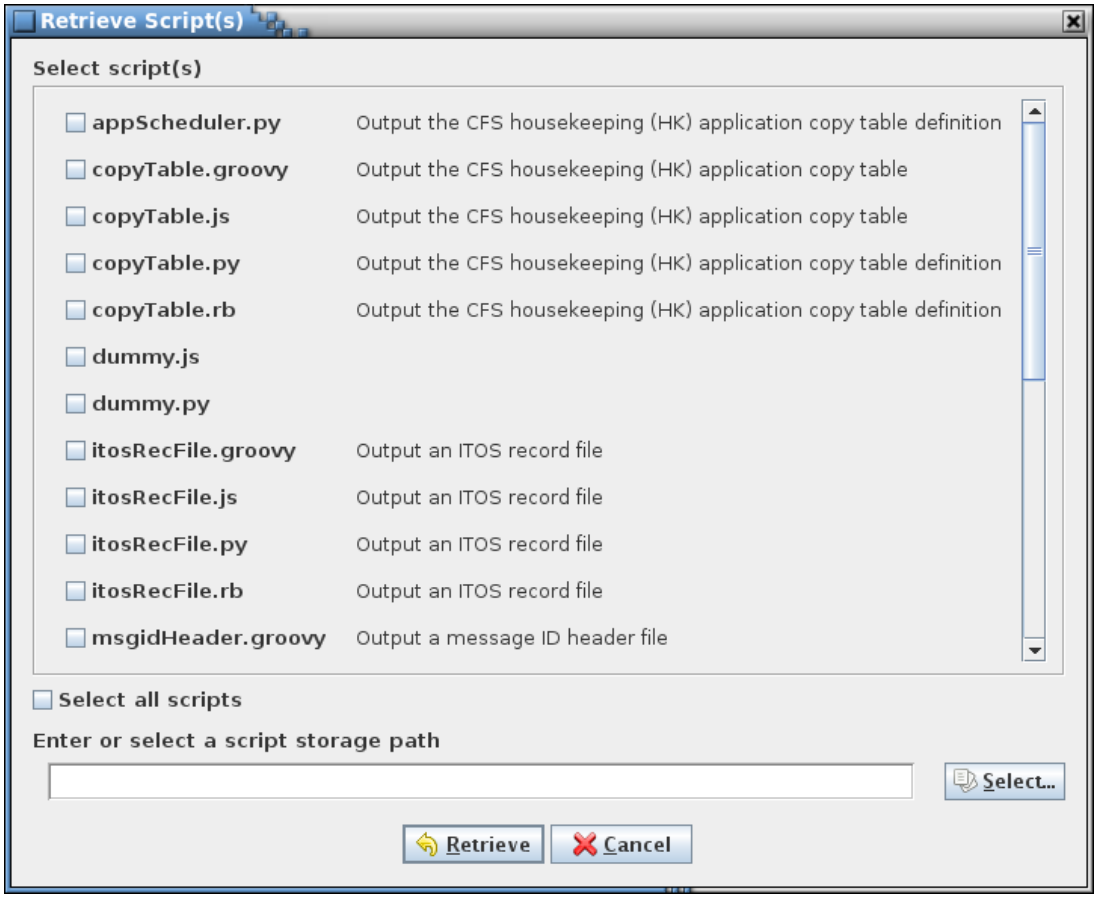


Figure 86. Retrieve Script(s) dialog

The dialog displays a list of the stored script (or other text) files. Using the check boxes, select one or more files. The **Select all scripts** check box is used to alternately select and deselect all of the individual script check boxes. Choose a folder in which to save the retrieved file(s), then press the **Retrieve** button to extract a copy of the file(s) from the project into the selected folder. Select the **Cancel** button to exit dialog without retrieving any files.

When a file is stored the application first searches it for the first line containing the text "description:". The search ignores case, so any combination of upper and lower case characters constitutes a match. If found, the remaining text on the same line in the file (sans any leading or trailing white space character(s)) is stored with the file as its description. This is the description that appears alongside the file name in the dialog. If no match is found then the description text in the dialog is blank.

4.9.5.5 Delete

Selecting the **Delete** command causes the Delete Script(s) dialog to appear (Figure 87). This command allows for deleting script (or other text files) stored in the project database using the **Store** command (see paragraph 4.9.5.3). This command is enabled only for a user with read/write or administrator access.

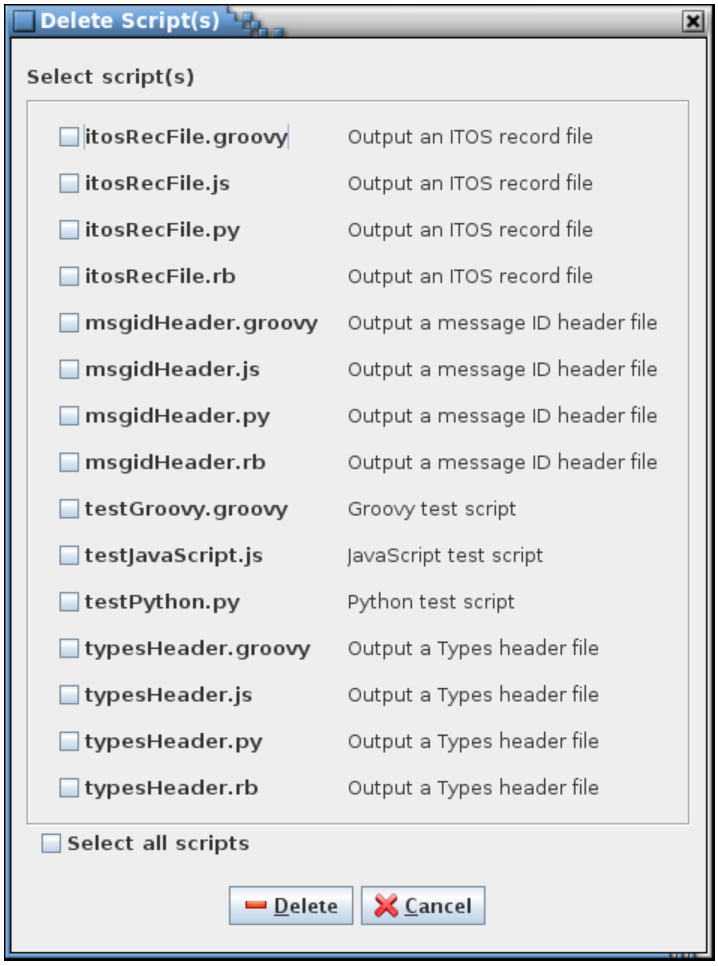


Figure 87. Delete Script(s) dialog

The dialog displays a list of the stored script (or other text) files. Using the check boxes, select one or more files. The **Select all scripts** check box is used to alternately select and deselect all of the individual script check boxes. Press the **Delete** button to delete the file(s) from the project. A confirmation dialog first appears; select **Okay** to continue with the script deletion, or **Cancel** to exit the dialog without deleting a script. Select the **Cancel** button to exit the dialog without deleting any files.

4.9.5.6 Search

The script **Search** dialog provides a means of searching for a specified text string within the scripts stored in the project database (see Figure 88). Case sensitivity for the search is governed by the **Ignore text case** check box.

The **Allow regular expression** check box, when checked, allows the use of a regular expression to define the search pattern in the search text field. A regular expression can be formulated to find multiple matching conditions (for example, the search for **a.c** would match any string that has a single character between the characters 'a' and 'c'). Information regarding the use of regular expressions is beyond the scope of this document; however, resources and tutorials can be found online. When unchecked, the search text is matched as typed in the search text field.

Enter the search text in the input field and select the **Search** button. The search results are displayed in the table at the bottom of the search dialog. The first column, **Script**, shows the name of the script, and the second column, **Line number**, provides the line number in the script where a match is found. The last column, **Context**, displays the line from the script containing the search text, with the search text

highlighted. Leading and trailing white space characters are removed from the context cells' text prior to display.

The search text field uses auto-completion to fill in the search string. The previous search strings (those for the event log, table, and script) are remembered, including those from previous sessions. The number of remembered search strings can be changed via the Preferences dalog, and defaults to 30. Case sensitivity for auto-completion is based on the **Ignore text case** check box selection state.

Another search can be performed by altering the search text and selecting the **Search** button again. The search results can be output to a file or printer by selecting the **Print** button. To exit the search dialog select the **Close** button.

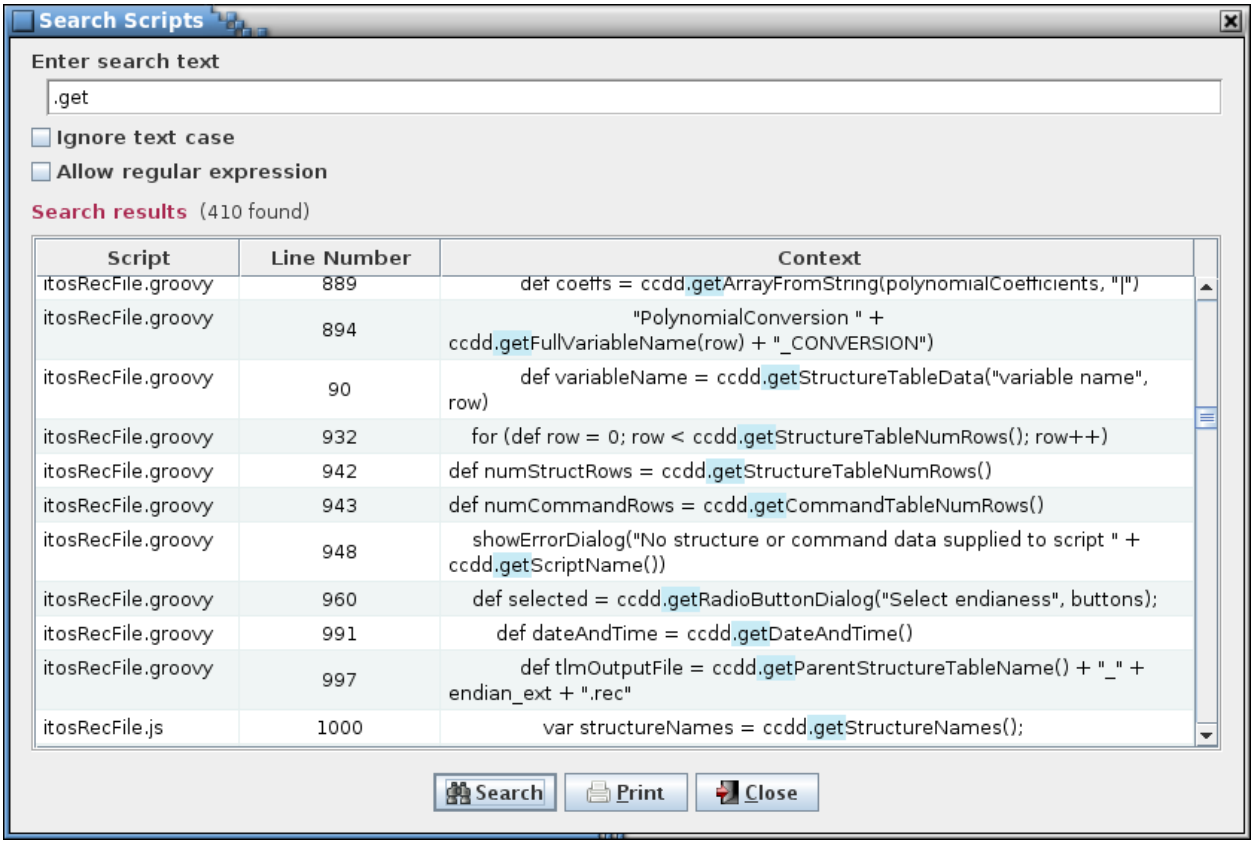


Figure 88. Script search dialog

4.9.6 Help

4.9.6.1 Guide

The **Guide** command displays a copy of this user's guide in PDF format. The file `CCDD_Users_guide.pdf` must be in the same folder in which the `CCDD.jar` file is located in order for the user's guide to be opened using this command.

4.9.6.2 About

Selecting the **About** menu item brings up an informational dialog (see Figure 89) providing the application's version number and date, version numbers for Java, PostgreSQL, JDBC, and Jetty that are in use, and the copyright notice. Also displayed is the list of available scripting languages and associated scripting engines, if any, and their respective version information.

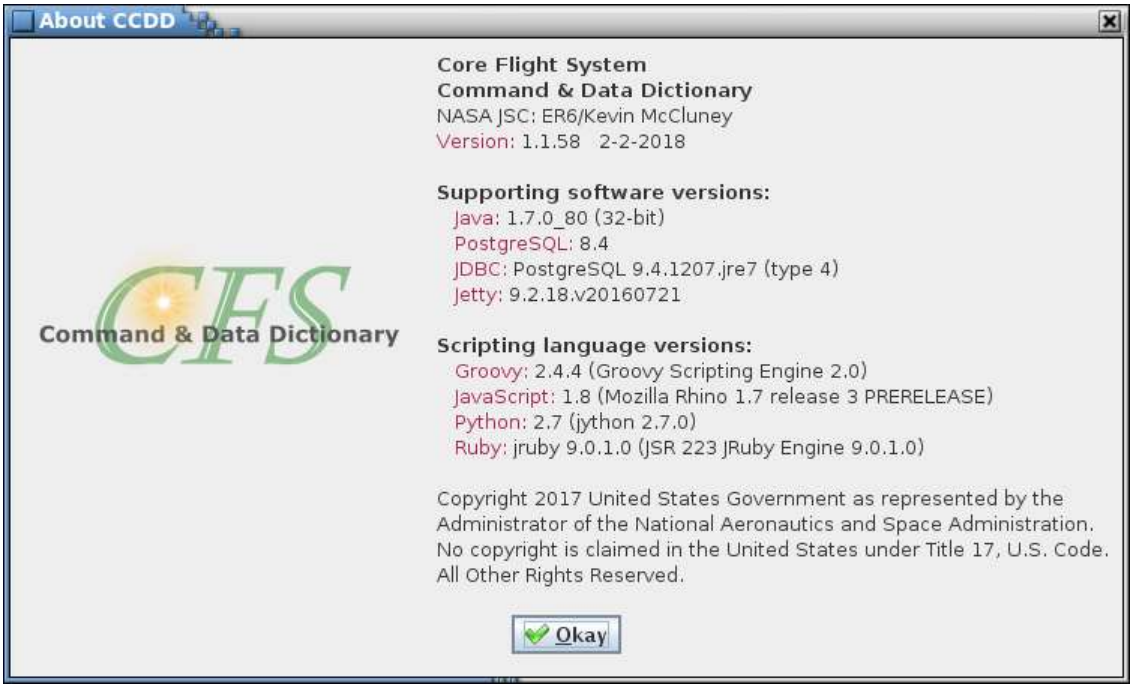


Figure 89. About dialog

4.10 Scripts

The CCDD application's script interface is the mean by which a project's data, stored in the database, is made available for manipulation by the user, primarily for formatting the data to create output files. CCDD supports the use of JVM-based scripting languages. Five of these languages, JavaScript, Python, Ruby, Groovy, and Scala, have been tested with the CCDD application, though any of the other compliant scripting languages should work as well. A language must be installed before it can be used by CCDD; paragraph 4.1 provides details on the library files required for using each of the five tested scripting languages. The About dialog (see paragraph 4.9.6.2 and Figure 89) displays a list of the installed scripting languages. Examples of the use of scripts to produce output files include creation of:

- C header files for CFS applications
- CFS Housekeeping copy table
- ITOS record and display files

Scripts may be executed from within the application (see paragraphs 4.9.5.1 and 4.9.5.2) or from the command line (see Table 1 and paragraph 4.10.5).

The scripts have access to the project data via a set of script data access methods written in Java. Additional methods are provided for displaying dialog boxes (both output and input), opening and writing to an output file, and making direct queries to the database. The methods are called from within a script using the method name and, dependent on the language, prepended by the class name **`ccdd`** or **`ccdds`**:

`<ccdd or ccdds.>methodName(arguments…)`

where *methodName* is the name of the script data access method (function) and *arguments…* are the parameters required by the particular method. **`ccdd`** is a reference to the non-static version of the script data access class, whereas **`ccdds`** is a static reference to the non-static class' methods. JavaScript scripts require the non-static reference in Java 8, but can use either in Java 7. Ruby scripts require the non-static reference, but Python and Groovy scripts can use either. Scala scripts must use the static

reference; however, the class name is not used in the script. Details on the script data access methods are provided in Table 10.

In order to access these methods the script requires that the data access class (non-static or static version) be imported; the import statement format is dependent on the scripting language. The following paragraphs show the import statement required to be included in the script file for each of the tested scripting languages, as well as an example of using the script data access methods. For each scripting language the example accomplishes the same result and assumes one or more structure tables are associated with the script (see paragraph 4.9.5.1 for information on associating scripts with data tables). First, the script opens an output file names "myFileName". Then the names of the structures present in the structure table(s) supplied to the script are stored in an array named "structNames". A loop is then performed to write each structure's name to the output file. Finally, the output file is closed and the script terminates, returning control to the CCDD application. A status message is written to the event log to indicate script completion.

If an error occurs, preventing successful script completion, an entry is made in the event log displaying the script name(s) and provides details on the cause of the error. The amount of detail provided depends on the scripting language. This can be improved by the use of exception catching in the script. The syntax is language dependent, but in general one or more sections of the script code is encompassed by a try-catch statement (usually the main portion and not any functions). An exception, caused by an error condition in the script, is caught. Data, such as the execution trace or variable values, can be included in the text that is returned to CCDD as the cause of the failure, which is then included in the event log entry. Information on the specific syntax is given in the following paragraphs.

4.10.1 JavaScript

JavaScript script files must end with the extension ".js". The JavaScript script must contain the following lines at or near the top of the file (this allows the script to work with both JavaScript 'Rhino' (Java 7 and earlier) and 'Nashorn' (Java 8 and later)):

```
try
{
    load("nashorn:mozilla_compat.js");
}
catch (e)
{
}
importClass(Packages.CCDD.CcddScriptDataAccessHandler);
```

If enhanced error logging is desired then encompass the script code with a `try-catch` statement as shown below. The `throw` call output can be replaced or have other text appended if desired.

```
import traceback
try
{
   # Main script steps
   .
   .
   .
}
catch (err)
{
   throw err.name + " " + err.message + " " + err.stack;
}
```

The following is the example script described earlier in this section, written in JavaScript:

```
// Import the script data access method class
```

```
try
{
    load("nashorn:mozilla_compat.js");
}
catch (e)
{
}
importClass(Packages.CCDD.CcddScriptDataAccessHandler);

// Open the output file
var file = ccdd.openOutputFile("myFileName");

// Get the array of structure names
var structNames = ccdd.getStructureTableNames();

// Step through each name found
for (var index = 0; index < structNames.length; index++)
{
// Write the structure name to the output file
ccdd.writeToFileLn(file,
                   "structNames["
                   + index
                   + "] = "
                   + structNames[index]);
}

// Close the output file
ccdd.closeFile(file);
```

4.10.2 Python

Python script files must end with the extension ".py". The Python script must contain the following line at or near the top of the file:

```
from CCDD import CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a `try-except` statement as shown below. The `raise` call output can be replaced or have other text appended if desired.

```
import traceback

try:
   # Main script steps
   .
   .
   .
except:
   raise Exception(traceback.format_exec())
```

The following is the example script described earlier in this section, written in Python:

```
# Import the script data access method class
from CCDD import CcddScriptDataAccessHandler

# Open the output file
file = ccdd.openOutputFile("myFileName")

# Get the array of structure names
structNames = ccdd.getStructureTableNames()

# Step through each name found
```

```
for index in range(len(structNames)):

   # Write the structure name to the output file
   ccdd.writeToFileLn(file, "structNames[" + str(index) + "] = " +
                      structNames[index])

# Close the output file
ccdd.closeFile(file)
```

4.10.2.1 Calling other scripts

It may be desirable for the main Python script called by the association to in turn call another Python script. In order for the 'child' script to access the script data access methods the following can be done:

- Create a folder representing the Python script package.
- Place the Python scripts in the package folder. The child script(s) must be placed in the package folder, but the main script does not.
- Create the file `__init__.py` in the package folder. In this file add an import statement for each child script in the form '`from child import *`' where *`child`* is the child script name (minus the `.py` extension).
- In the child scripts add '`from CCDD import CcddScriptDataAccessHandlerStatic as ccdd`'. Notice that this must be a reference to the static version of the script data access method class. Access method call format in the child script are identical to those in the main script (i.e., `ccdd.`*`methodName`*`()`).
- In the main script add '`import sys`' and '`sys.path.append("`*`path to the package folder`*`")`', then '`import `*`package folder name`*'. The system path must include the package folder's location and the update must occur before importing the package.
- To call a function in a child script from the main script use the format '*`package folder name.child function name`*`()`'.

4.10.3 Ruby

Ruby script files must end with the extension ".rb". The Ruby script must contain the following line at or near the top of the file:

```
java_import Java::CCDD.CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a `begin-rescue` statement as shown below. The `raise` call output can be replaced or have other text appended if desired.

```
begin
   # Main script steps
   .
   .
   .
rescue => err
   raise err.message + "; " + err.backtrace.join("; ")
end
```

The following is the example script described earlier in this section, written in Ruby:

```
# Import the script data access method class
java_import Java::CCDD.CcddScriptDataAccessHandler

# Open the output file
file = $ccdd.openOutputFile("myFileName")
```

```
# Get the array of structure names
structNames = $ccdd.getStructureTableNames()

index = 0

# Step through each structure name
structNames.each do |name|

    # Write the structure name to the output file
    $ccdd.writeToFileLn(file, "structNames[#{index}] = #{name}")

    index += 1

end

# Close the output file
$ccdd.closeFile(file)
```

4.10.4 Groovy

Groovy script files must end with the extension ".groovy". The Groovy script must contain the following line at or near the top of the file:

```
import CCDD.CcddScriptDataAccessHandler
```

If enhanced error logging is desired then encompass the script code with a `try-catch` statement as shown below. The `throw` call output can be replaced or have other text appended if desired.

```
try
{
   # Main script steps
   .
   .
   .
}
catch (Exception err)
{
   throw new Exception(err.message + "; " + err.getStackTrace())
}
```

The following is the example script described earlier in this section, written in Groovy:

```
// Import the script data access method class
import CCDD.CcddScriptDataAccessHandler

// Open the output file
def file = ccdd.openOutputFile("myFileName")

// Get the array of structure names
def structNames = ccdd.getStructureTableNames()

// Step through each name found
for (def index = 0; index < structNames.length; index++)
{
    // Write the structure name to the output file
    ccdd.writeToFileLn(file,
                       "structNames[" +
                       index +
                       "] = " +
                       structNames[index])
}
```

```
// Close the output file
ccdd.closeFile(file)
```

4.10.5 Scala

Scala script files must end with the extension ".scala". The Scala script must contain the following line at or near the top of the file:

```
import CCDD.CcddScriptDataAccessHandlerStatic._
```

If enhanced error logging is desired then encompass the script code with a `try-catch` statement as shown below. The `throw` call output can be replaced or have other text appended if desired.

```
try
{
   # Main script steps
   .
   .
   .
}
catch
{
   case err: Exception => throw new Exception(err.message + "; " +
   err.getStackTrace().mkString("; "))
}
```

The following is the example script described earlier in this section, written in Scala:

```
// Import the script data access method class
import CCDD.CcddScriptDataAccessHandlerStatic._

// Open the output file
var file = openOutputFile("myFileName")

// Get the array of structure names
var structNames = getStructureTableNames()

// Step through each name found
for (index <- 0 to structNames.length - 1)
{
    // Write the structure name to the output file
    writeToFileLn(file,
                  "structNames[" +
                  index +
                  "] = " +
                  structNames(index))
}

// Close the output file
closeFile(file)
```

4.10.6 Command line execution

The CCDD command line option, `execute`, allows running scripts without use of the GUI. The script file and data table association must be specified on the command line. The command format is:

```
<script_name[:table[+table2[+...[+tableN]]][;...]]>
```

Groups can be used in place of, or along with tables. Each referenced group name must be preceded by 'Group:' in order to be recognized as a group. For example:

```
script_name:Group:group_name
```

The project database, host, user, and password (if required) command line options must be specified as part of the `execute` option in order to access the project's database. If not specified, the last project database, user, and host accessed by the application in the most recent session is used. The script name must include its file path if the script is not located within the folder from which the CCDD application is executed. If multiple scripts are provided in the same `execute` command then the individual associations must be separated by a semi-colon (;) and the entire string containing the associations for that `execute` command must be bounded by single or double quotes. Multiple `execute` commands in the same command line command can be used as well to execute multiple script associations; the format for each is as described above. If multiple script associations are specified then these are run serially in the order they appear in the command line command.

Even though the GUI is not displayed, the event log is generated and all events (success, fail, command, and status events) are written to the log file. Information, warning, and error dialogs are not displayed; instead the text for these dialogs is output to the standard output (information) and standard error (warning and error) streams. Dialogs within a script requiring user input, however, are displayed, and script execution pauses until the dialog is dealt with.

When script execution completes the CCDD application terminates. The application returns a status indicating if the scripts executed successfully: 0 if all script execution succeeded, or 1 if any script did not complete successfully.

Following are examples of running scripts from the command line. Note that in these examples CCDD is an alias that executes the application with all the necessary class paths, etc. (see paragraph 4.1). The first example demonstrates executing the script *myScript* with no associated tables:

```
CCDD -project myProject -host localHost -user myUser -password myPassword
  -execute myScript
```

The next example executes *myScript* using the data from the table *myTable* (and its child tables, if applicable):

```
CCDD -project myProject -host 192.168.1.1 -port 5432 -user myUser -
  password myPassword -execute myScript:myTable
```

The third example executes *myScript* using the data from the tables *myTable1* and *myTable2* (and their child tables, if applicable):

```
CCDD -project myProject -user myUser -password myPassword -execute
  myScript:myTable1+myTable2
```

The last example executes *myScript1* using the data from the tables *myTable1* and *myTable2* (and their child tables, if applicable), then executes the script *myScript2* using the data from the table *myTable3* (and its child tables, if applicable):

```
CCDD -password myPassword -execute
  myScript1:myTable1+myTable2,myScript2:myTable3
```

4.10.7 XTCE export

The XTCE export dialog (paragraph 4.9.3.8.4) provides a means of exporting table data into a file in XTCE format. The XTCE schema allows for some interpretation as to how data is parsed to the output and provides for many features not covered by the export methods within the CCDD application, so the content of the CCDD-produced XTCE file may not be as desired. A script can be created to produce the conversion; however, there are certain operations performed in the internal code that would be difficult to replicate in a script - for example, the conversion of the command header structure table to the

equivalent command metadata required by the XTCE schema. To make XTCE conversion via a script easier a hybrid of the internal code and script code can be used.

The internal code has "hooks" that allow for several of the key conversion methods to be replaced with code in a script. These hooks are activated when exporting via the XTCE export dialog by selecting the **Use external methods** check box, or when the script access method **xtceExport** is called from a script.

Table 8 below provides a brief description of each of the XTCE export methods that can be overridden by a script function. The script must use the same function name as shown in the table, and the **Input(s)** and **Output** columns describe the parameters that are passed to and expected from the script functions. Figure 90 displays a flow chart of the methods' calling sequence. If the script doesn't have the function to replace a method then the internal method is used by default.

The user can elect to arrange the conversion process in a manner other than by the predefined methods. Assuming the xtceExport() script data access method or the export dialog is used as the conversion initiator, the **addSpaceSystemParameters** and **addSpaceSystemCommands** must be the entry points in the script.

Internal Method / Script Function Name	Description	Input(s)	Output
addSpaceSystemHeader	Add the Header element to the space system. For the root space system the AuthorSet and NoteSet elements are automatically created and populated with the user, project, creation, and endianness information. The script data access method **xtceAddSpaceSystemHeader** can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space system reference String; Classification attribute String: Validation status attribute String: Version attribute String: Creation time and date	
createTelemetryMetadata	Create the TelemetryMetaData element within the specified space system. The script data access method **xtceCreateTelemetryMetadata** can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space system reference	
addSpaceSystemParameters	Overall parameter assignment handler; steps through each row of data in the structure table and calls functions that assign variables to the ParameterTypeSet, ParameterSet, and ContainerSet elements. If the application ID is provided, the table is a root structure, and it has a variable with a data type referencing the telemetry header table, then a BaseContainer element is created with a RestrictionCriteria element set to the application ID. The script data access method **xtceAddSpaceSystemParameters** can be called instead to use the internal method.	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system to which the table belongs String: Table name String[][]: Array containing the table's data int: Variable (parameter) name column index int: Parameter data type column index int: Parameter array size column index int: Parameter bit length column index	

Internal Method / Script Function Name	Description	Input(s)	Output
		int: Parameter enumeration column index; -1 if no the table has no enumeration column int: Parameter description column index; -1 if no the table has no description column int: Parameter units column index; -1 if no the table has no units column int: Minimum parameter value column index; -1 if no the table has no minimum column int: Maximum parameter value column index; -1 if no the table has no maximum column boolean: true if this table represents the telemetry header or one of its descendants String: telemetry header table system path; null or blank is none boolean: true if the table is a root structure table String: Name of the telemetry header application ID data field String: Telemetry header application ID	
addParameterAndType	Add a structure table variable, if it has a primitive data type, to the ParameterTypeSet and ParameterSet elements. The script data access method **xtceAddParameterAndType** can be called instead to use the internal method.	ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian	

Internal Method / Script Function Name	Description	Input(s)	Output
		boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system reference String: Parameter name String: Parameter primitive data type String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or blank if not a bit-wise parameter String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units String: Minimum parameter value String: Maximum parameter value String: Parameter description int: Size, in characters, of a string parameter; ignored if not a string or character	
setParameterDataType	Add a structure table variable as a parameter type in the ParameterTypeSet element. The parameter type is based on the variable's data type. An array variable generates two entries, one to describe the array type, and a second to describe the data type of the array elements. The script data access method **xtceSetParameterDataType** can be called instead to use the internal method.	ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Telemetry header table name SpaceSystemType: Space system	

Internal Method / Script Function Name	Description	Input(s)	Output
		String: Parameter name; null to not specify String: Data type; null to not specify String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or empty if not a bit-wise parameter String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Parameter description; null to not specify int: Size, in characters, of a string parameter; ignored if not a string or character	
addParameterSequenceEntry	Add the structure table variables as EntryList entries in a SequenceContainer element within a ContainerSet element. The script data access method **xtceAddParameterSequenceEntry** can be called instead to use the internal method.	ObjectFactory: Object factory reference String: Telemetry header table name SpaceSystemType: Reference to the space system to which the parameter belongs String: Parameter name String: Data type String: Array size	boolean: true if the parameter's data type references the telemetry header or one of its descendants; otherwise return the flag status unchanged

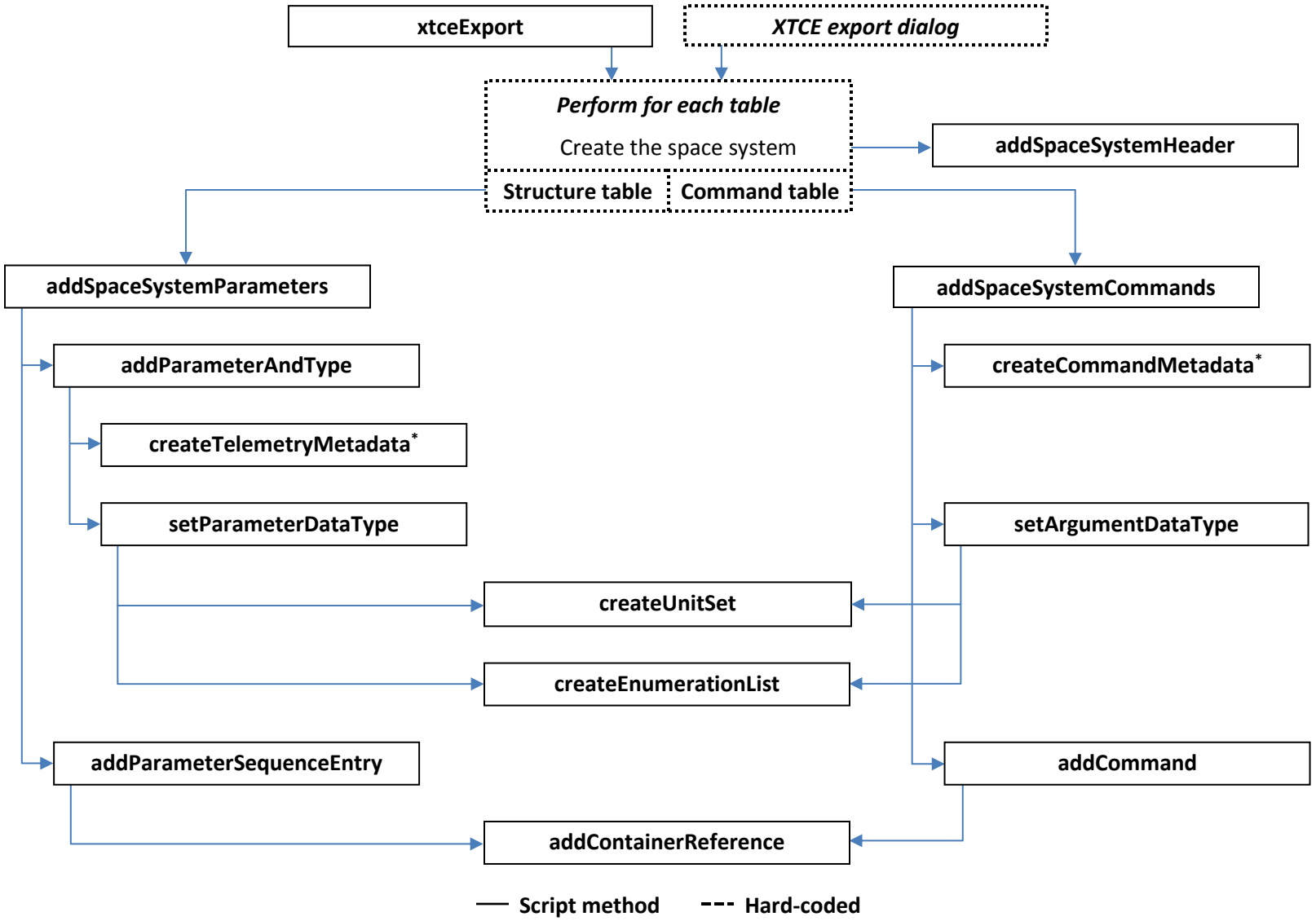
Internal Method / Script Function Name	Description	Input(s)	Output
		EntryListType: Reference to the entry list into which to place the parameter (for a primitive data type) or container (for a structure data type) reference boolean: true if this table represents the telemetry header or one of its descendants	
createCommandMetadata	Create the CommandMetaData element within the specified space system. The script data access method **xtceCreateCommandMetadata** can be called instead to use the internal method.	ObjectFactory: Object factory reference SpaceSystemType: Space reference	
addSpaceSystemCommands	Overall command assignment handler; steps through each row of data in the command table, parses the command's aruments, and calls functions that assign arguments to the ArgumentTypeSet element. For each command in the table a function is called to create the MetaCommand element, which describes the command, within the MetaCommandSet. The script data access method **xtceAddSpaceSystemCommands** can be called instead to use the internal method.	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Command header table name AssoctiatedColumns[]: Array of AssociatedColumns class instances that have the associated command argument column indices SpaceSystemType: Space system reference String[][]: Table data array int: Command name column index int: Command code column index int: Command description column index	

Internal Method / Script Function Name	Description	Input(s)	Output
		boolean: true if this table represents the command header String: Command header table system path String: Command code name String: Name of the application ID String: Application ID	
addCommand	Add a MetaCommand element to the MetaCommandSet. Each MetaCommand element has a BaseMetaCommand (if the command header table is defined), which uses ArgumentAssignment elements to set the application ID and command function code. The ArgumentList and CommandContainer are populated with the command argument information. The script data access method **xtceAddCommand** can be called instead to use the internal method.	JAXBElement<SpaceSystemType>: Top-level space system element reference ObjectFactory: Object factory reference String: Command header table name SpaceSystemType: Space system reference String: Command name String: Command code name String: Command code String: Name of the application ID String: Application ID boolean: true if this table represents the command header String: Command header table system path String[]: Array of command argument names String[]: Array of of command argument data types String[]: Array of of command argument array sizes; the array item is null or	

Internal Method / Script Function Name	Description	Input(s)	Output
		blank if the corresponding argument isn't an array String: Description of the command	
setArgumentDataType	Add a command table command argument as an argument type in the ArgumentTypeSet element. The argument type is based on the argument's data type. An array variable generates two entries, one to describe the array type, and a second to describe the data type of the array elements. The script data access method **xtceSetArgumentDataType** can be called instead to use the internal method.	boolean: true if the data is big endian; false for little endian boolean: true if the telemetry and command headers are big endian String: Command header table name SpaceSystemType: Space system reference String: Command argument name; null to not specify String: Command argument data type; null to not specify String: Command argument array size; null or blank if the argument isn't an array String: Command argument bit length String: Command argument enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Command argument units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Command argument description ; null to not specify	NameDescriptionType: Command description of the type corresponding to the primitive data type with the specified attributes set

Internal Method / Script Function Name	Description	Input(s)	Output
		int: String size in bytes; ignored if the command argument does not have a string data type	
addContainerReference	Generic function used by both the telemetry and command functions to add a container reference to the specified EntryList element. The script data access method **xtceAddContainerReference** can be called instead to use the internal method.	ObjectFactory: object factory reference EntryListType: Reference to the telemetry or command entry list into which to place the parameter or parameter array container reference(s) String: Parameter name String: Data type String: Parameter array size; null or blank if the parameter isn't an array	
createUnitSet	Use the provided units information to create a UnitSet element. The script data access method **xtceCreateUnitSet** can be called instead to use the internal method.	ObjectFactory: object factory reference String: Parameter or command argument units; null to not specify	UnitSet: Unit set for the supplied units string; an empty unit set if no units are supplied
createEnumerationList	Use the provided enumeration information to create an EnumerationList element. The script data access method **xtceCreateEnumerationList** can be called instead to use the internal method.	ObjectFactory: object factory reference SpaceSystemType: Space system reference String: Enumeration in the format <enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]	EnumerationList: Enumeration list for the supplied enumeration string

Table 8. Overridable XTCE export methods



* *Called once per space system, and only if a telemetry/command parameter exists in the table*

Figure 90. XTCE export script method flow

4.10.8 Data access methods

Table 10 provides details on each of the project data access methods available for use in the scripts. The first column is the method name. The scripts are automatically assigned a variable, `ccdd`, which references the class containing the data access methods. When calling one of the access methods from a script the method name must be preceded by `ccdd` (or `ccdds` if using the static methods) for JavaScript, Python, and Groovy, and `$ccdd` for Ruby; for Scala only the method name is used. The second column is a short description of the access method. The third column in the table gives the method input parameter type(s) and description(s), if any. The fourth column gives the output type and description, if any. The last column indicates the applicability of the method to the project data. See Table 9 for the definitions of the applicability codes.

Certain methods require that the table type be supplied as a parameter. Convenience methods are provided in these cases for the Structure and Command table types. In place of supplying the table type as a parameter the method name incorporates the table type. For example, the method `getTableData` has accompanying convenience methods `getCommandTableData` and `getStructureTableData`.

Code	Method Applicability
O	Method returns information with respect to only those tables associated with the script. This includes every method with a row number input
A	Method returns information for any or all tables, not just those associated with the script
S	Method returns information from the telemetry or application schedulers, so is not dependent on the associated tables
N	Method is not table related, so is not dependent on the associated tables

Table 9. Data access method applicability code definitions

Method Name	Description	Input(s)	Output	*
closeFile	Close the specified output file	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method		N
formatArrayIndex	Convert an integer array containing the size of each array dimension into a string in the format [#]<[#]<...>>	int[]: Array of integers containing the size of each array dimension	String: Array size in the format [#]<[#]<...>>	N
getApplicationMessageDefinitionTable	Get the application scheduler message definition table		String[]: Array containing the message definition table information	S
getApplicationNames	Get the array containing the groups that represent CFS applications		String[]: Array containing names of the groups that represent CFS applications	A
getApplicationScheduleDefinitionTable	Get the specified entry in the application scheduler schedule definition table	int: Row index for the entry in the schedule definition table	String[][]: Array containing the specified entry in the schedule definition table	S
getApplicationScheduleDefinitionTableDefines	Get the array of defined parameters for the schedule definition table		String[]:Two-dimensional array containing the defined parameters	S

Method Name	Description	Input(s)	Output	*
getArrayFromString	Divide the supplied string into a two-dimensional array (columns and rows) using the supplied separator characters or strings, and trim any leading or trailing white space characters from each array member	String: String to separate into an array String: Character string to use to delineate the separation point(s) between columns. The separator is eliminated from the array members String: Character string to use to delineate the separation point(s) between rows. The separator is eliminated from the array members. Use null if only one row is supplied	String[][]: Two-dimensional array representing the substrings in the supplied text after being parsed using the separator; returns null if the input text is empty	N
getArrayFromString	Divide the supplied string into an array using the supplied separator character or string, and trim any leading or trailing white space characters from each array member	String: String to separate into an array String: Character string to use to delineate the separation point(s) between columns. The separator is eliminated from the array members	String[]: Array representing the substrings in the supplied text after being parsed using the separator; returns null if the input text is empty	N
getArrayIndexFromSize	Get the integer array containing the size of each array dimension from the supplied array size string	String: Array size in the format [#]<[#]<...>> or #<,#<...>>	int[]: Array of integers containing the size of each array dimension	N
getAssociatedGroupNames	Get the array of group names referenced in the script association		String[]: Array containing the group names referenced in the script association; empty array if no groups are referenced	O

Method Name	Description	Input(s)	Output	*
getBaseDataType	Get the base type for the specified data type	String: Name of the primitive data type	String: Base type for the specified data type; returns null if the data type doesn't exist or isn't a primitive type	N
getCDataType	Get the C type for the specified data type	String: Name of the primitive data type	String: C type for the specified data type; returns null if the data type doesn't exist or isn't a primitive type	N
getCheckBoxDialog	Display a dialog containing one or more check boxes. The user must press the Okay button to accept the check box input(s), or Cancel to close the dialog without accepting the input	String: Text to display above the check box(es) String[][]: Array containing the text and optional descriptions for the radio buttons to display in the dialog	boolean[]: An array containing the status for the check box(es) if the Okay button is pressed; returns null if no check box information is supplied or if the Cancel button is pressed	N
getCommandArgArraySize	Get the argument array size for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument array size for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O

Method Name	Description	Input(s)	Output	*
getCommandArgArraySizeWithMacros	Get the argument array size for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument array size for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O
getCommandArgBitLength	Get the argument bit length for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument bit length for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O
getCommandArgBitLengthWithMacros	Get the argument bit length for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument bit length for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O

Method Name	Description	Input(s)	Output	*
getCommandArgByColumnName	Get the argument value (as a string) for the column belonging to the specified command argument at the specified row in the command data, with any macro name replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index String: Name of the argument column for which the value is requested	String: Argument value (as a string) for the column belonging to the specified command argument at the specified row in the command data; null if the argument number, row index, or column name is invalid	O
getCommandArgByColumnNameWithMacros	Get the argument value (as a string) for the column belonging to the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index String: Name of the argument column for which the value is requested	String: Argument value (as a string) for the column belonging to the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number, row index, or column name is invalid	O
getCommandArgColumnNames	Get the array of column names belonging to the specified command argument at the specified row in the command data	int: Command argument index. The first argument is 0 int: Table data row index	String[]: Array of column names belonging to the specified command argument at the specified row in the command data; null if the argument number or row index is invalid	O

Method Name	Description	Input(s)	Output	*
getCommandArgDataType	Get the argument data type for the specified command argument at the specified row in the command data	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument data type for the specified command argument at the specified row in the command data; null if the argument number or row index is invalid	O
getCommandArgEnumeration	Get the argument enumeration for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument enumeration for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O
getCommandArgEnumerationWithMacros	Get the argument enumeration for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument enumeration for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O
getCommandArgMaximum	Get the argument maximum (as a string) for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument maximum value (as a string) for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O

Method Name	Description	Input(s)	Output	*
getCommandArgMaximumWithMacros	Get the argument maximum (as a string) for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument maximum value (as a string) for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O
getCommandArgMinimum	Get the argument minimum value (as a string) for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument minimum value (as a string) for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O
getCommandArgMinimumWithMacros	Get the argument minimum value (as a string) for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument minimum value (as a string) for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O

Method Name	Description	Input(s)	Output	*
getCommandArgName	Get the argument name for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument name for the specified command argument at the specified row in the command data, with any macro replaced by its corresponding value; null if the argument number or row index is invalid	O
getCommandArgNameWithMacros	Get the argument name for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place	int: Command argument index. The first argument is 0 int: Table data row index	String: Argument name for the specified command argument at the specified row in the command data, with any embedded macro(s) left in place; null if the argument number or row index is invalid	O
getCommandCode	Get the command code (as a string) at the specified row in the command data	int: Table data row index	String: Command code (as a string) at the specified row in the command data; null if the row index is invalid	O
getCommandInformation	Get an array containing the name, code, table, and argument name(s) for every command in the project database		String[][]: Array containing the name, code, table, and argument name(s) for every command. The array is sorted by command name; if the same then by command code; if the same then by table name	A

Method Name	Description	Input(s)	Output	*
getCommandName	Get the command name at the specified row in the command data	int: Table data row index	String: Command name at the specified row in the command data; null if the row index is invalid	O
getCommandTableColumnNames	Get the column names for the table referenced on the specified row of the command table data	int: Command table data row index	String[]: Array containing the names of the columns of the command table referenced in the specified row of the command table data	O
getCommandTableData	Get the command table data at the row and column indicated, with any macro replaced by its corresponding value. The column is specified by name and is not case sensitive. Convenience method for getTableData that assumes the table type is "command"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified command table's array at the row and column name provided, with any macro replaced by its corresponding value; returns null if an instance of the command table type doesn't exist	O
getCommandTableDataFieldValues	Get the data field value for all command tables that have the specified data field	String: Data field name	String: Array of command table names and the data field value; returns an empty array if the field name is invalid (i.e., no command table has the data field)	O

Method Name	Description	Input(s)	Output	*
getCommandTableDataWithMacros	Get the command table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive. Convenience method for getTableDataWithMacros that assumes the table type is "command"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified command table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the command table type doesn't exist	O
getCommandTableNameByRow	Get the command table name to which the specified row's data belongs. Convenience method for getTableNameByRow that assumes the table type is "command"	int: Table data row index	String: Command table name to which the current row's parameter belongs; returns a blank if an instance of the command table type or the row doesn't exist	O
getCommandTableNames	Get the array of all command table names in the table data. Convenience method for getTableNames that specifies the table type as "command"		String[]: Array of all command table names; returns an empty array if an instance of the command table type doesn't exist	O
getCommandTableNumRows	Get the number of rows of data in the command table. Convenience method for getTableNumRows that assumes the table type is "command"		int: Number of rows of data in the table for the table type "command"; return -1 if an instance of the command table type doesn't exist	O

Method Name	Description	Input(s)	Output	*
getCommandTypeNameByRow	Get the table type name referenced in the specified row of the command table type data. Convenience method for getTypeNameByRow that specifies the table type as "command". The data for all command types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	int: Table data row index	String: Command table type name to which the current row's parameter belongs; returns a blank if an instance of the command table type or the row doesn't exist	O
getCopyTableColumnNames	Get the copy table column names		String[]: Array containing the copy table column names	S
getCopyTableEntries	Get the copy table for the messages of the specified data stream	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String: Name of the message ID name data field (e.g., 'Message ID name') boolean: true to combine memory copy calls for consecutive variables in the copy table boolean: false to retain any macros in the variable names; true to replace any macros with their corresponding values	String[][]: Array containing the copy table entries; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S

Method Name	Description	Input(s)	Output	*
getCopyTableEntries	Get the copy table for the messages of the specified data stream	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String[][]: Array containing string array entries giving the structure table path+name and the table's associated message ID name boolean: true to combine memory copy calls for consecutive variables in the copy table boolean: false to retain any macros in the variable names; true to replace any macros with their corresponding values	String[][]: Array containing the copy table entries; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S

Method Name	Description	Input(s)	Output	*
getCopyTableEntriesWithMacros	Get the copy table for the messages of the specified data stream. Any macro embedded in a variable name is left in place	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String: Name of the message ID name data field (e.g., 'Message ID name') boolean: true to combine memory copy calls for consecutive variables in the copy table	String[][]: Array containing the copy table entries with any macro embedded in a variable name left in place; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getCopyTableEntriesWithMacros	Get the copy table for the messages of the specified data stream. Any macro embedded in a variable name is left in place	String: Data stream name int: Size of the message header in bytes. For example, the CCSDS header size is 12 String[][]: Array containing string array entries giving the structure table path+name and the table's associated message ID name boolean: true to combine memory copy calls for consecutive variables in the copy table	String[][]: Array containing the copy table entries with any macro embedded in a variable name left in place; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S

Method Name	Description	Input(s)	Output	*
getDatabaseQuery	Perform a query on the currently open database	String: PostgreSQL-compatible database query statement	String[][]: Two-dimensional array representing the rows and columns of data returned by the database query; returns null if the query produces an error, or an empty array if there are no results	N
getDataStreamNames	Get a string array containing all of the data stream names in the project		String[]: Array containing the unique data stream names	N
getDataTypeDefinitions	Get the array containing the user-defined data type names and their corresponding C-language, size (in bytes), and base data type values		String[][]: Array where each row contains a user-defined data type name and its corresponding C-language, size (in bytes), and base data type values	N
getDataTypeSizeInBits	Get the number of bits for the specified data type	String: Name of the structure or primitive data type	int: Number of bits required to store the data type; returns 0 if the data type doesn't exist	N
getDataTypeSizeInBytes	Get the number of bytes for the specified data type	String: Name of the structure or primitive data type	int: Number of bytes required to store the data type; returns 0 if the data type doesn't exist	N

Method Name	Description	Input(s)	Output	*
getDateAndTime	Get the current time and date in the form: *dow mon dd hh*:*mm*:*ss zzz yyyy* where: *dow* is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat) *mon* is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) *dd* is the day of the month (01 through 31), as two decimal digits *hh* is the hour of the day (00 through 23), as two decimal digits *mm* is the minute within the hour (00 through 59), as two decimal digits *ss* is the second within the minute (00 through 61, as two decimal digits *zzz* is the time zone (and may reflect daylight saving time) *yyyy* is the year, as four decimal digits		String: Current date and time	N
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by underscores, and with the data types removed	int: Table data row index	String: The variable's full path and name with each variable in the path separated by an underscore, and with the data types removed; returns a blank if the row is invalid	O

Method Name	Description	Input(s)	Output	*
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the supplied separator character(s)	int: Table data row index String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s), and with the data types removed; returns a blank if the row is invalid	O
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the supplied separator character(s). Data types may be excluded or retained, based on the input flag. If retained, the data types and variable names are separated by the supplied separator character(s)	int: Table data row index String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the row is invalid	O
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s) and with the data types removed. In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2)	String: Path to the variable in the format *rootTable*[,*structureDataType1*.*variable1*[,*structureDataType2*.*variable2*[,...]]] String: Name of the variable in the format *primitiveDataType*.*variable* String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s) and with the data types removed; returns a blank if the variable path + name doesn't exist in the project database	A

Method Name	Description	Input(s)	Output	*
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s). In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2). Data types may be excluded or retained, based on the input flag	String: Path to the variable in the format *rootTable*[,*structureDataType1*.*variable1*[,*structureDataType2*.*variable2*[,...]]] String: Name of the variable in the format *primitiveDataType*.*variable* String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the variable path + name doesn't exist in the project database	A
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s), and with the data types removed. In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2)	String: Variable path + name in the format *rootTable*[,*structureDataType1*.*variable1*[,*structureDataType2*.*variable2*[,...]]],*primitiveDataType*.*variable* String: Character(s) to place between the variable path members	String: The variable's full path and name with each variable in the path separated by the specified separator character(s), and with the data types removed; returns a blank if the variable path + name doesn't exist in the project database	A

Method Name	Description	Input(s)	Output	*
getFullVariableName	Get a variable's full name which includes the variables in the structure path separated by the specified separator character(s). In case there are any array member variable names in the full name, replace left square brackets with # underscores and remove right square brackets (example: a[0],b[2] becomes a_0separatorb_2). Data types may be excluded or retained, based on the input flag. Any macro embedded in the variable name is expanded	String: Variable path + name in the format *rootTable*[,*structureDataType1*.*variable1*[,*structureDataType2*.*variable2*[,...]]],*primitiveDataType*.*variable* String: Character(s) to place between the variable path members boolean: true to exclude the data types from the path + name String: Character(s) to place between the data types and variable names	String: The variable's full path and name with each variable in the path separated by the specified separator character(s); returns a blank if the variable path + name doesn't exist in the project database. Any macro embedded in the variable name is expanded	A
getFullVariableNameRaw	Get the full name of the variable in the specified row of the structure datain the application's native format, which includes the variables in the structure path separated by commas, and with the data type and variable names separated by periods	int: Table data row index	String: The variable's full path and name with each variable in the path separated by a comma, and with each data type and variable name separated by a period; returns a blank if the row is invalid	O
getGroupDataFieldDescription	Get the description of the data field for the specified group's specified data field	String: Group name String: Data field name	String: Data field's description; returns a blank if the group name or data field name is invalid	N

Method Name	Description	Input(s)	Output	*
getGroupDataFieldNames	Get the name(s) of the data field(s) associated with the specified group	String: Name of the group to which the field is a member	String: Array of the data field names associated with the specified group; returns an empty array if the group name is invalid or the group has no data fields	N
getGroupDataFieldValue	Get the contents of the data field for the specified table's specified data field	String: Table name, including the path if this table references a structure String: Data field name	String: Data field's value; returns a blank if the table type, table name, or data field name is invalid	N
getGroupDescription	Get the description for the specified group	String: Group name	String: Description for the specified group; blank if the group has no description or the group doesn't exist	N
getGroupFields	Get the data field information for the specified group	String: Group name	String[][]: Array containing the data field information for the specified group; an empty array if the group has no data fields, or the group doesn't exist. Each row in the array describes a single data field in the format: field name, description, size, input type, required (true or false), applicability, value	N
getGroupNames	Get an array of all group names	boolean: true to get only the groups that represent a CFS application; false to get all groups	String[]: Array containing the group names (application groups only if the input flag is true); returns an empty array if no groups exist	N

Method Name	Description	Input(s)	Output	*
getGroupTables	Get an array containing the table members, including the member table ancestor tables, for the specified group	String: Group name	String[]: Array containing the table members for the specified group; an empty array if the group has no table members, or the group doesn't exist	A
getInputDialog	Display a dialog for receiving text input. The user must select Okay to accept the input, or Cancel to close the dialog without accepting the input	String: Text label to display beside the input text field	String: The text entered in the dialog input field if the Okay button is pressed; returns null if no text or white space is entered, or if the Cancel button is pressed	N

Method Name	Description	Input(s)	Output	*
getITOSEncodedDataType	Convert a primitive data type into its ITOS encoded form	String: Name of the data type; e.g., uint16, double String: ITOS encoding type: **SINGLE_CHAR** to get the single character encoding (e.g., "I" for any integer type) **TWO_CHAR** to get the encoding character with the data type size (e.g., "I4" for a 4-byte integer) **BIG_ENDIAN** to get the encoding as big endian **BIG_ENDIAN_SWAP** to get the encoding as a big endian with byte swapping **LITTLE_ENDIAN** to get the encoding as little endian **LITTLE_ENDIAN_SWAP** to get the encoding as a little endian with byte swapping	String: ITOS encoded form of the data type in the format requested; returns the data type, unmodified, if the data type is a table (i.e., it's a structure), or null if the data type is unrecognized. Example: a data type of "int32" and ITOS encoding type of **LITTLE_ENDIAN** returns "I12345678"	N
getITOSLimitName	Get the ITOS limit name based on the supplied index value	int: 0 = redLow, 1 = yellowLow, 2 = yellowHigh, 3 = redHigh	String: ITOS limit name ("redLow", "yellowLow", "yellowHigh", or "redHigh"); returns blank if the index is invalid	N

Method Name	Description	Input(s)	Output	*
getLinkApplicationNames	Get the array containing the application name data field values associated with the specified link's variable members. Each application name is listed only once in the array	String: Name of the application name data field	String[]: Array containing the contents of the specified application name data field associated with each of the tables referenced by the link's variable members	N
getLinkDescription	Return the description for the specified link; returns a blank if the link doesn't exist or the link has no description	String: Data stream name String: Link name	String: Link description; returns a blank if the data stream or link don't exist, or the link has no description	N
getLinkRate	Return the sample rate for the specified link; returns a blank if the link doesn't exist	String: Data stream name String: Link name	String: Text representation of the sample rate, in samples per second, of the specified link. For rates equal to or faster than 1 sample per second the string represents a whole number; for rates slower than 1 sample per second the string is in the form *number of samples / number of seconds*; returns a blank if the data stream or link don't exist	N
getLongestString	Get the character length of the longest string in the supplied string array	String[]: Array of strings Integer: Initial minimum width; null to use zero as the minimum	int: Character length of the longest string in the supplied array; null if an input is invalid	N

Method Name	Description	Input(s)	Output	*
getLongestStrings	Get the character length of the longest string for each column in the supplied string array	String[][]: Array of string arrays Integer[]: Initial minimum widths; null to use zero as the minimums	Integer[]:Character length of the longest string in each column of the supplied array; null if any of the inputs is invalid	N
getMacroDefinitions	Get the array containing the macro names and their corresponding values		String[][]: Array where each row contains a macro names and its corresponding value	N
getMessageOwnersIDsAndNames	Get an array containing every message name and its corresponding ID, and the owning entity from every table cell, data field (table or group), and telemetry message. Message names and IDs are determined by the input data type assigned to the table column or data field		String[][]: Two-dimensional array containing every message name and its corresponding ID, and the owning entity, sorted by owner name. Each row in the array is an array in the form [owner name], [message name], [message ID]. The owner name is preceded by 'Group:' if the owner is a group, and by "Tlm:' if the owner is a telemetry message	A
getNumberOfTimeSlots	Get the number of time slots for the scheduler definition table		int: Number of time slots for the scheduler definition table table	N

Method Name	Description	Input(s)	Output	*
getNumCommandArguments	Get the number of arguments associated with the command table type at the specified row in the command data	int: Row index	int: Number of arguments associated with the command table type at the specified row in the command data	O
getNumCommandArguments	Get the number of arguments associated with the specified command table type	String: Table type (case insensitive)	int: Number of arguments associated with the specified command table type; -1 if the table type is invalid	O
getOutputPath	Get the script output file path set via the program preferences dialog or command line		String: Script output file path; blank if no path has been set	N

Method Name	Description	Input(s)	Output	*
getPathByRow	Get the path to which the specified row's data belongs with any embedded macro replaced by its corresponding value	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: The path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: *rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]*	O

Method Name	Description	Input(s)	Output	*
getPathByRowWithMacros	Get the path to which the specified row's data belongs with any embedded macro(s) left in place	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: The path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: *rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]*	O
getProject	Get the project's name		String: Name of the project	N
getProjectDataFieldDescription	Get the description of the data field for the specified project data field	String: Data field name	String: Data field's description; returns a blank if the project data field name is invalid	N

Method Name	Description	Input(s)	Output	*
getProjectDataFieldNames	Get the name(s) of the data field(s) associated with the project		String: Array of the data field names associated with the project; returns an empty array if the project has no data fields	N
getProjectDataFieldValue	Get the contents of the data field for the specified project data field	String: Data field name	String: Data field's value; returns a blank if the project data field name is invalid	N
getProjectDescription	Get the project's description		String: Description of the project	N
getProjectFields	Get the data field information for the project		String[][]: Array containing the data field information for the project; an empty array if the project has no data fields. Each row in the array describes a single data field in the format: field name, description, size, input type, required (true or false), applicability, value	N
getPrototypeName	Get the name of the prototype table for the specified table	String: Table name	String: The name of the prototype table for the specified table	N

Method Name	Description	Input(s)	Output	*
getRadioButtonDialog	Display a dialog containing radio buttons. The radio buttons are mutually exclusive; only one can be selected at a time. The user must press the Okay button to accept the radio button input, or Cancel to close the dialog without accepting the input	String: Text to display above the radio buttons String[][]: Array containing the text and optional descriptions for the radio buttons to display in the dialog	String: The text for the selected radio button if the Okay button is pressed; returns null if no radio button is selected or if the Cancel button is pressed	N
getRootStructureTableNames	Get the name(s) of the root structure table(s). Convenience method for getRootTableNames that assumes the table type is "structure"		String[]: Array containing the root structure table names; returns an empty array if an instance of the structure table type doesn't exist	O
getRootTableNames	Get the name(s) of the root table(s) for the supplied table type. Note that only structure tables can have child tables so using this method for non-structure tables returns the same list of tables as getTableNames(typeName)	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	String[]: Array containing the root table names for the type specified; returns an empty array if an instance of the table type doesn't exist	O
getScriptName	Get the name of the script file being executed		String: Script file name	N
getStructureArraySize	Get the variable array size at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable array size at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O

Method Name	Description	Input(s)	Output	*
getStructureArraySizeWithMacros	Get the variable array size at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable array size at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureBitLength	Get the variable bit length at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable bit length at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureBitLengthWithMacros	Get the variable bit length at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable bit length at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureDataByVariableName	Get the data from the specified "Structure" table in the specified column for the row with the specified variable name, with any macro replaced by its corresponding value. Convenience method for getTableDataByColumnName that assumes the table type is "Structure" and the variable name column is "Variable Name"	String: Full table path, which includes the parent table name and the data type + variable name pairs String: Variable name String: Column name (case insensitive)	String: Contents of the table defined by the table path, variable name, and column name specified; returns null if an instance of the table type, the column name, or the variable name doesn't exist	O

Method Name	Description	Input(s)	Output	*
getStructureDataByVariableNameWithMacros	Get the data from the specified "Structure" table in the specified column for the row with the specified variable name, with any macro name(s) left in place. Convenience method getTableDataByColumnName that assumes the table type is "Structure" and the variable name column is "Variable Name"	String: Full table path, which includes the parent table name and the data type + variable name pairs String: Variable name String: Column name (case insensitive)	String: Contents of the table defined by the table path, variable name, and column name specified, with any macro name(s) left in place; returns null if an instance of the table type, the column name, or the variable name doesn't exist	O
getStructureDataType	Get the variable data type at the specified row in the structure data	int: Table data row index	String: Variable data type at the specified row in the structure data; null if the row index is invalid	O
getStructureDescription	Get the variable description at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable description at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid or no column has the 'Units' input type	O
getStructureDescriptionWithMacros	Get the variable description at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable description at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid or no column has the 'Units' input type	O

Method Name	Description	Input(s)	Output	*
getStructureEnumerations	Get the variable enumeration(s) at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Array containing the variable enumeration(s) at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureEnumerationsWithMacros	Get the variable enumeration(s) at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Array containing the variable enumeration(s) at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O
getStructureParentRowByChildRow	Get the row index in the structure data for the first entry associated with the parent structure of the entry on the specified row of the structure data	int: Table data row index	int: The row index in the structure data for the first entry associated with the parent structure of the entry on the specified row of the structure data	O

Method Name	Description	Input(s)	Output	*
getStructurePathByRow	Get the structure path to which the specified row's data belongs with any embedded macro replaced by its corresponding value. Convenience method that assumes the table type is "structure"	int: Table data row index	String: The structure path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: *rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]*	O

Method Name	Description	Input(s)	Output	*
getStructurePathByRowWithMacros	Get the structure path to which the specified row's data belongs with any embedded macro(s) left in place. Convenience method that assumes the table type is "structure"	int: Table data row index	String: The structure path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. For structure tables the root name is followed by a comma and then the parent structure and variable name(s) that define(s) the table's path. Each parent and its associated variable name are separated by a period. Each parent/variable pair in the path is separated by a comma. The format is: *rootTable[,structureDataType1.variable1[,structureDataType2.variable2[...]]]*	O
getStructureRates	Get the variable rate(s) at the specified row in the structure data	int: Table data row index	String: Array containing the variable rate(s) at the specified row in the structure data; null if the row index is invalid	O

Method Name	Description	Input(s)	Output	*
getStructureTablesByReferenceOrder	Get an array containing the names of the prototype structures in the order in which they are referenced; that is, the structure array is arranged so that a structure appears in the array prior to a structure that references it		String[]: Array containing the names of the prototype structures in the order in which they are referenced	O
getStructureTableColumnNames	Get the column names for the table referenced on the specified row of the structure table data	int: Structure table data row index	String[]: Array containing the names of the columns of the structure table referenced in the specified row of the structure table data	O
getStructureTableData	Get the structure table data at the row and column indicated, with any macro replaced by its corresponding value. The column is specified by name and is not case sensitive. Convenience method for getTableData that assumes the table type is "structure"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified structure table's array at the row and column name provided, with any macro replaced by its corresponding value; returns null if an instance of the structure table type doesn't exist	O
getStructureTableDataFieldValues	Get the data field value for all structure tables that have the specified data field	String: Data field name	String: Array of structure table names and the data field value; returns an empty array if the field name is invalid (i.e., no structure table has the data field)	O

Method Name	Description	Input(s)	Output	*
getStructureTableDataWithMacros	Get the structure table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive. Convenience method for getTableDataWithMacros that assumes the table type is "structure"	String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified structure table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the structure table type doesn't exist	O
getStructureTableITOSPathByRow	Get the structure path to which the specified row's data belongs, formatted for use in an ITOS record statement, and with any macro name replaced by its corresponding value	int: Table data row index	String: The path to the current row's parameter formatted for use in an ITOS record statement and with any macro name replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a period and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by an underscore. The format is: *rootTable[.variable1[.variable2[...]]]*	O

Method Name	Description	Input(s)	Output	*
getStructureTableITOSPathByRowWithMacros	Get the structure path to which the specified row's data belongs, formatted for use in an ITOS record statement, and with any macro name(s) left in place	int: Table data row index	String: The path to the current row's parameter formatted for use in an ITOS record statement and with any macro name(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a period and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by an underscore. The format is: *rootTable[.variable1[.variable2[...]]]*	O
getStructureTableNameByRow	Get the prototype structure table name to which the specified row's data belongs. Convenience method for getTableNameByRow that assumes the table type is "structure"	int: Table data row index	String: Prototype structure table name to which the current row's parameter belongs; returns a blank if an instance of the structure table type or the row doesn't exist	O
getStructureTableNames	Get array of all prototype structure table names referenced in the table data. Convenience method that specifies the table type as "structure"		String[]: Array of all prototype structure table names; returns an empty array if an instance of the structure table type doesn't exist	O

Method Name	Description	Input(s)	Output	*
getStructureTableNumRows	Get the number of rows of data in the structure table. Convenience method for getTableNumRows that assumes the table type is "structure"		int: Number of rows of data in the table for the table type "structure"; return -1 if an instance of the structure table type doesn't exist	O
getStructureTablePaths	Get array of all structure table names, including paths for child structure tables, referenced in the table data. Convenience method that specifies the table type as "structure"		String[]: Array of all structure table names, including paths for child structure tables; returns an empty array if an instance of the structure table type doesn't exist	O
getStructureTableVariablePathByRow	Get the structure path to which the specified row's data belongs, showing only the root structure and variable names and with any embedded macro replaced by its corresponding value. This format is used when referencing a structure table's data fields	int: Table data row index	String: The path to the current row's parameter with any embedded macro replaced by its corresponding value; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a comma and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by a comma. The format is: *rootTable[,variable1[,variable2[...]]]*	O

Method Name	Description	Input(s)	Output	*
getStructureTableVariablePathByRowWithMacros	Get the structure path to which the specified row's data belongs and with any embedded macro(s) left in place, showing only the root structure and variable names. This format is used when referencing a structure table's data fields	int: Table data row index	String: The path to the current row's parameter with any embedded macro(s) left in place; returns a blank if an instance of the table type doesn't exist or the row number is invalid. The path starts with the root table name. The root name is followed by a comma and then the variable name(s) that define(s) the table's path. Each variable in the path is separated by a comma. The format is: *rootTable[,variable1[,variable2[...]]]*	O
getStructureTypeNameByRow	Get the table type name referenced in the specified row of the structure table type data. Convenience method for or getTypeNameByRow that specifies the table type as "structure". The data for all structure types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	int: Table data row index	String: Structure table type name to which the current row's parameter belongs; returns a blank if an instance of the structure table type or the row doesn't exist	O

Method Name	Description	Input(s)	Output	*
getStructureUnits	Get the variable units at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable units at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid or no column has the 'Description' input type	O
getStructureUnitsWithMacros	Get the variable units at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable units at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid or no column has the 'Description' input type	O
getStructureVariableName	Get the variable name at the specified row in the structure data, with any macro name replaced by its corresponding value	int: Table data row index	String: Variable name at the specified row in the structure data, with any macro name replaced by its corresponding value; null if the row index is invalid	O
getStructureVariableNameWithMacros	Get the variable name at the specified row in the structure data, with any embedded macro(s) left in place	int: Table data row index	String: Variable name at the specified row in the structure data, with any embedded macro(s) left in place; null if the row index is invalid	O

Method Name	Description	Input(s)	Output	*
getTableColumnNames	Get the table column names for the table referenced on the specified row of the table data for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String[]: Array containing the names of the columns of the table type referenced in the specified row of the type's table data	O
getTableColumnNamesByType	Get the table column names for the table type specified	String: Table type name	String[]: Array containing the names of the columns of the table type specified	A
getTableData	Get the data at the row and column indicated, with any macro replaced by its corresponding value, for the table type specified. The column is specified by name	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified table's array at the row and column name provided with any macro replaced by its corresponding value; returns null if an instance of the table type, the column name, or the row doesn't exist	O

Method Name	Description	Input(s)	Output	*
getTableDataByColumnName	Get the data from the a table in the specified column for the row in the matching column name that contains the matching name, with any macro name replaced by its corresponding value	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Full table path String: Name of the column containing that matching name (case insensitive) String: Text to match in the matching column - this determines the row. The first row in the matching column that matches the matching name determines the row used to retrieve the data value String: Name of the column from which to retrieve the data value (case insensitive)	String: Contents of the table defined by the table type, table path, matching column name, matching name, and data column name specified, with any macro name replaced by its corresponding value; returns null if an instance of the table type, the matching column, the data column, or the matching name doesn't exist	O

Method Name	Description	Input(s)	Output	*
getTableDataByColumnNameWithMacros	Get the data from the a table in the specified column for the row in the matching column name that contains the matching name, with any macro name(s) left in place	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Full table path String: Name of the column containing that matching name (case insensitive) String: Text to match in the matching column - this determines the row. The first row in the matching column that matches the matching name determines the row used to retrieve the data value String: Name of the column from which to retrieve the data value (case insensitive)	String: Contents of the table defined by the table type, table path, matching column name, matching name, and data column name specified, with any macro name(s) left in place; returns null if an instance of the table type, the matching column, the data column, or the matching name doesn't exist	O
getTableDataFieldDescription	Get the description of the data field for the specified table's specified data field	String: Table name, including the path if this table references a structure String: Data field name	String: Data field's description; returns a blank if the table name or data field name is invalid	A

Method Name	Description	Input(s)	Output	*
getTableDataFieldNames	Get the name(s) of the data field(s) associated with the specified table	String: Name of the table, including the path if this table references a structure, to which the field is a member	String: Array of the data field names associated with the specified table; returns an empty array if the table name is invalid or the table has no data fields	A
getTableDataFieldValue	Get the contents of the data field for the specified table's specified data field	String: Table name, including the path if this table references a structure String: Data field name	String: Data field's value; returns a blank if the table name or data field name is invalid	A
getTableDataFieldValues	Get the data field value for all tables of the specified type that have the specified data field	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command". null to include tables of any type String: Data field name	String: Array of table names of the specified type and the data field value; returns an empty array if the field name is invalid (i.e., no table has the data field)	O
getTableDataFieldValues	Get the data field value for all tables that have the specified data field	String: Data field name	String: Array of table names and the data field value; returns an empty array if the field name is invalid (i.e., no table has the data field)	O

Method Name	Description	Input(s)	Output	*
getTableDataWithMacros	Get the table data at the row and column indicated, with any macro name(s) left in place. The column is specified by name and is not case sensitive	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Table column name (case insensitive) int: Table data row index	String: Contents of the specified table's array at the row and column name provided, with any macro name(s) left in place; returns null if an instance of the table type, the column name, or the row doesn't exist	O
getTableDescription	Get the description of the specified table	String: Table name, including the full path for child structure tables	String: Description of the specified table; returns a blank the table doesn't exist	A
getTableDescriptionByRow	Get the description of the table at the row indicated for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Table name for the specified table type to which the current row's parameter belongs; returns a blank if an instance of the table type or the row doesn't exist	O

Method Name	Description	Input(s)	Output	*
getTableNameByRow	Get the prototype table name for the type specified to which the specified row's parameter belongs	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Prototype table name to which the current row's parameter belongs; return a blank if an instance of the table type or the row doesn't exist	O
getTableNames	Get array of all table names referenced in the table data of the specified table type	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" boolean: true to return only the prototype name for any child structures; false to include the full path for child structures	String[]: Array of all table names, with paths for child structure tables excluded based on the input flag, represented by the table type; returns an empty array if an instance of the table type doesn't exist	O

Method Name	Description	Input(s)	Output	*
getTableNames	Get array of all table names, including paths for child structure tables, referenced in the table data for all table types		String[]: Array of all table names, including paths for child structure tables, referenced in the table data; empty array if no tables exists in the data	O
getTableNames	Get array of all table names, including paths for child structure tables, referenced in the table data of the specified table type	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	String[]: Array of all table names, including paths for child structure tables, represented by the table type (prototype names for child structures); returns an empty array if an instance of the table type doesn't exist	O
getTableNumRows	Get the number of rows of data for all table types		int: Number of rows of data for all table types; return 0 if there is no table data	O
getTableNumRows	Get the number of rows of for the table type specified	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command"	int: Number of rows of data in the table for the table type specified; return -1 if an instance of the table type doesn't exist	O

Method Name	Description	Input(s)	Output	*
getTelemetryMessageIDs	Get the copy table message ID names and their corresponding ID values for the specified data stream	String: Data stream name	String: Array containing the copy table message ID names and ID values; returns blank if there are no entries for the specified data stream or if data stream name is invalid	S
getTypeDataFieldDescription	Get the description of the data field for the specified table type's specified data field	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Data field name	String: Data field's description; returns a blank if the table type name or data field name is invalid	A
getTypeDataFieldNames	Get the name(s) of the data field(s) associated with the specified table type	String: Name of the table type to which the field is a member	String: Array of the data field names associated with the specified table type; returns an empty array if the table type name is invalid or the table type has no data fields	A

Method Name	Description	Input(s)	Output	*
getTypeDataFieldValue	Get the contents of the data field for the specified table type's specified data field	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" String: Data field name	String: Data field's value; returns a blank if the table type name or data field name is invalid	A
getTypeNameByRow	Get the table type name referenced in the specified row of the specified table type data. Multiple structure (and command) types are allowed. The data for all structure (command) types are combined. This method provides the means to retrieve the specific table type to which the row data belongs	String: Table type (case insensitive). All structure table types are combined and are referenced by the type name "Structure", and all command table types are combined and are referenced by the type name "Command" int: Table data row index	String: Table type name to which the current row's parameter belongs; returns a blank if an instance of the table type or the row doesn't exist	O
getTypeNameByTable	Get the the table type name for the specified table	String: Name of the table. For a child structure this includes the path	String: Type name for the specified table	O
getUser	Get the name of the user executing the script		String: Name of the user executing the script	N

Method Name	Description	Input(s)	Output	*
getVariableLinks	Get the array of link names to which the specified variable belongs	String: Variable path and name	String[]: Array containing the links to which the specified variable is a member; returns an empty array if the variable does not belong to a link	N
getVariableOffset	Get the byte offset of the specified variable relative to its parent structure. The variable's path, including parent structure and variable name, is used to verify that the specified target has been located; i.e., not another variable with the same name	String: Parent structure name of the variable being checked String: A comma separated string of each data type and variable name of each variable in the current search path	int: The byte offset to the target variable relative to its parent structure; returns -1 if the parent-variable path combination is invalid	A
getVariablePaths	Get an array containing the path to each parent structure and its variables		String[][]: Array containing the path for each structure variable. The root structures are sorted alphabetically. The variables are displayed in the order of appearance within the structure (parent or child). Any macro is replaced by its corresponding value	A
isArrayMember	Check if the supplied variable name represents an array member	Object: Variable name	boolean: true if the variable name is an array member	N
isDataTypeCharacter	Determine if the supplied data type is a character or string	String: Name of the data type	boolean: true if the supplied data type is a character or string; false otherwise	N

Method Name	Description	Input(s)	Output	*
isDataTypeFloat	Determine if the supplied data type is a float or double	String: Name of the data type	boolean: true if the supplied data type is a float or double; false otherwise	N
isDataTypeInteger	Determine if the supplied data type is a signed or unsigned integer	String: Name of the data type	boolean: true if the supplied data type is a signed or unsigned integer; false otherwise	N
isDataTypePrimitive	Determine if the supplied data type is a primitive type	String: Name of the data type	boolean: true if the supplied data type is a primitive; false otherwise	N
isDataTypeString	Determine if the supplied data type is a character string	String: Name of the data type	boolean: true if the supplied data type is a character string; false otherwise	N
isDataTypeUnsignedInt	Determine if the supplied data type is an unsigned integer	String: Name of the data type	boolean: true if the supplied data type is an unsigned integer; false otherwise	N
isStructureShared	Determine if the specified structure is referenced by more than one root structure	String: Prototype name of the structure to check	boolean: true if the specified structure is referenced by more than one root structure; false otherwise	N
openOutputFile	Open the specified file for writing. The PrintWriter object that is returned is used by the file writing methods to specify the output file	String: Output file path + name	PrintWriter: PrintWriter object; returns null if the file could not be opened	N

Method Name	Description	Input(s)	Output	*
parseEnumerationParameters	Divide the supplied enumeration string into the values and labels. The enumeration value/label separator character and the enumerated pair separator character are automatically determined. Any leading or trailing white space characters are removed from each array member	String: Enumeration in the format <enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]	String[][]: Two-dimensional array representing the enumeration parameters ; returns null if the input text is empty or the enumeration separator characters cannot be determined	N
showErrorDialog	Display an error dialog showing the supplied text. The dialog's header and icon indicate that the text describes an error condition. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N
showInformationDialog	Display an informational dialog showing the supplied text. The dialog's header and icon indicate that the text describes information useful to the user; e.g., script status. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N
showWarningDialog	Display a warning dialog showing the supplied text. The dialog's header and icon indicate that the text describes a warning condition. The Okay button must be pressed before the script can continue	String: Text to display in the dialog box		N

Method Name	Description	Input(s)	Output	*
writeFailLogEntry	Write the supplied text to the event log with an event type of 'Fail. The log message is prepended with "[script: *scriptFileName*] " where *scriptFileName* is the file name of the script generating the message request	String: Text to output to the event log		N
writeStatusLogEntry	Write the supplied text to the event log with an event type of 'Status. The log message is prepended with "[script: *scriptFileName*] " where *scriptFileName* is the file name of the script generating the message request	String: Text to output to the event log		N
writeSuccessLogEntry	Write the supplied text to the event log with an event type of 'Success'. The log message is prepended with "[script: *scriptFileName*] " where *scriptFileName* is the file name of the script generating the message request	String: Text to output to the event log		N
writeToFile	Write the supplied text to the specified output file PrintWriter object	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Text to write to the output file		N

Method Name	Description	Input(s)	Output	*
writeToFileFormat	Write the supplied formatted text in the indicated format to the specified output file PrintWriter object	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Print format string to write to the output file Object…: variable list of arguments referenced by the format specifiers in the format string		N
writeToFileLn	Write the supplied text to the specified output file PrintWriter object and append a line feed character	PrintWriter: Output file PrintWriter object obtained from the openOutputFile method String: Text to write to the output file		N
xmlCleanSystemPath	Replace each invalid character with an underscore and move any leading underscores to the end of each path segment	String: System path in the form <</>path1</path2<...>>	String: Path with each invalid character replaced with an underscore and any leading underscores moved to the end of each path segment	N
xmlGetBaseDataType	Convert the primitive data type into the base equivalent used by the xtceSetParameterDataType() and xtceSetArgumentDataType() methods	String: Data type	BasePrimitiveDataType: Base primitive data type corresponding to the specified primitive data type; null if no match	N

Method Name	Description	Input(s)	Output	*
xtceAddCommand	Add a command to the command metadata set	SpaceSystemType: Space system reference String: Command name String: Command code String: Application ID boolean: true if this table represents the command header String: Command header table system path String[]: Array of command argument names String[]: Array of of command argument data types String[]: Array of of command argument array sizes; the list item is null or blank if the corresponding argument isn't an array String: Description of the command		N

Method Name	Description	Input(s)	Output	*
xtceAddContainerReference	Add a container reference(s) for the telemetry or command parameter or parameter array to the specified entry list	String: Reference to the telemetry or command entry list into which to place the parameter or parameter array container reference(s) String: Parameter name String: Data type String: Parameter array size; null or blank if the parameter isn't an array		N

Method Name	Description	Input(s)	Output	*
xtceAddParameterAndType	Add a parameter with a primitive data type to the parameter set and parameter type set	SpaceSystemType: Space system reference String: Parameter name String: Parameter primitive data type String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or blank if not a bit-wise parameter String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units String: Minimum parameter value String: Maximum parameter value String: Parameter description int: Size, in characters, of a string parameter; ignored if not a string or character		N

Method Name	Description	Input(s)	Output	*
xtceAddParameterSequenceEntry	Add the parameter to the sequence container entry list	SpaceSystemType: Reference to the space system to which the parameter belongs String: Parameter name String: Data type String: Array size EntryListType: Reference to the entry list into which to place the parameter (for a primitive data type) or container (for a structure data type) reference boolean: true if this table represents the telemetry header or one of its descendants	boolean: true if the parameter's data type references the telemetry header or one of its descendants; otherwise return the flag status unchanged	N

Method Name	Description	Input(s)	Output	*
xtceAddSpaceSystemCommands	Add the command(s) from a table to the specified space system	SpaceSystemType: Space system reference String[][]: Table data array int: Command name column index int: Command code column index int: Command description column index boolean: true if this table represents the command header String: Command header table system path String: Application ID		N
xtceAddSpaceSystemHeader	Set the space system header attributes	SpaceSystemType: Space system reference String: Classification attribute String: Validation status attribute String: Version attribute String: Export creation time and date		N

xtceAddSpaceSystemParameters	Add a structure table's parameters to the telemetry meta data	SpaceSystemType: Space system reference String: Table name String[][]: Array containing the table's data int: Variable (parameter) name column index int: Parameter data type column index int: Parameter array size column index int: Parameter bit length column index int: Parameter enumeration column index; -1 if no the table has no enumeration column int: Parameter description column index; -1 if no the table has no description column int: Parameter units column index; -1 if no the table has no units column int: Minimum parameter value column index; -1 if no the table has no minimum column int: Maximum parameter value column index; -1 if no the table has no maximum column		N

Method Name	Description	Input(s)	Output	*
		boolean: true if this table represents the telemetry header or one of its descendants String: Telemetry header table system path; null or blank is none boolean: true if the table is a root structure table String: Telemetry header application ID		
xtceCreateCommandMetadata	Create the space system command metadata	SpaceSystemType: Space system reference		N
xtceCreateEnumerationList	Build an enumeration list from the supplied enumeration string	SpaceSystemType: space system reference String: Enumeration in the format <enum value><enum value separator><enum label>[<enum value separator>...][<enum pair separator>...]	EnumerationList:	N
xtceCreateTelemetryMetadata		SpaceSystemType: Space system reference		N
xtceCreateUnitSet	Build a unit set from the supplied units string	String: Parameter or command argument units; null to not specify	UnitSet:	N

Method Name	Description	Input(s)	Output	*
xtceExport	Export the tables in XTCE XML format to the specified file. This is the main entry point when using a script association to perform the export. It calls the internal method to set up and parse the tables for export	String: Output file name boolean: true if the data is big endian boolean: true if the telemetry and command headers big endian String: Version attribute (for the space system headers) String: Validation status attribute (for the space system headers) String: First level classification attribute (for the space system headers) String: Second level classification attribute (for the space system headers) String: Third level classification attribute (for the space system headers)	boolean: true if an error occurred preventing exporting the project to the file	N

xtceSetArgumentDataType	Set the command argument data type and set the specified attributes	SpaceSystemType: Space system reference String: Command argument name; null to not specify String: Command argument data type; null to not specify String: Command argument array size; null or blank if the argument isn't an array String: Command argument bit length String: Command argument enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Command argument units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Command argument description ; null to not specify int: String size in bytes; ignored if the command argument does not have a string data type String: Text used to uniquely identify data types with the	NameDescriptionType: Command description of the type corresponding to the primitive data type with the specified attributes set	N

Method Name	Description	Input(s)	Output	*
		same name; blank if the data type has no name conflict		

Method Name	Description	Input(s)	Output	*
xtceSetParameterDataType	Create the telemetry parameter data type and set the specified attributes	SpaceSystemType: Space system reference String: Parameter name; null to not specify String: Data type; null to not specify String: Parameter array size; null or blank if the parameter isn't an array String: Parameter bit length; null or empty if not a bit-wise parameter String: Enumeration in the format <enum label>|<enum value>[|...][,...]; null to not specify String: Parameter units; null to not specify String: Minimum parameter value; null to not specify String: Maximum parameter value; null to not specify String: Parameter description; null to not specify int: Size, in characters, of a string parameter; ignored if not a string or character		N

Table 10. Script data access methods

Appendix A. Acronyms

CCDD	CFS Command & Data Dictionary
CCSDS	Consultative Committee for Space Data Systems
cFE	Core Flight Executive
CFS	Core Flight System
CPU	Central Processing Unit
CSV	comma-separated values
DBU	Database Backup
EDS	Electronic Data Sheet
GUI	Graphical User Interface
HK	Housekeeping
I/O	Input/Output
ID	Identifier
ITOS	Integrated Test and Operations System
JAR	Java Archive
JDBC	Java Database Connectivity
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
L&F	Look and Feel
OID	Object Identifier
OS	Operating System
PDF	Portable Document Format
PNG	Portable Network Graphics
SQL	Structured Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XML	Extensible Markup Language
XTCE	XML Telemetric and Command Exchange

Appendix B. Definitions

Array definition In a structure table, the row where the variable name and array size are specified.

Array member In a structure table, the rows following the array definition row (when arrays are expanded) that display the individual variables that belong to the array. The variable name begins with the array definition's variable name and has the array index, encased in square brackets, appended. The array member rows are displayed in ascending index order, starting with a zero index. The array size column for each member row displays the total number of members in the array.

Child table A structure table that is referenced as a data type for a variable in another structure table.

Data type A primitive or structure data type; see **Primitive type** and **Structure type**.

Database A collection of data within a PostgreSQL server organized as tables. A CCDD *project* is a PostgreSQL database representing the data for a CFS project.

Encoded type The byte order for primitive data types composed of two or more bytes. CCDD recognizes four encodings:

little endian	bytes are store with the least significant byte first.
big endian	bytes are stores with the least significant byte last.
little endian, swap	similar to little endian, except that each byte pair is reversed; applies only to integer and unsigned integer data types composed of one or more pairs of bytes.
big endian, swap	similar to big endian, except that each byte pair is reversed; applies only to integer and unsigned integer data types composed of one or more pairs of bytes.

Instance table A structure table that is a child of another structure table (the child's *parent* table).

Macro An alphanumeric string, bounded by special delimiter characters, that can be inserted into a data table cell to represent text defined by the user.

Parent table The structure table for which a table is an immediate descendant (child). The parent and root tables are the same if this table is a child of a root table.

Path Refers to a table path or variable path.

Primitive type A primitive data type is a basic data type (e.g., integer, float), as opposed to a structure, which is a group of primitive and/or structure data types. The primitive data types recognized by the CCDD application can be altered using the Data Type Manager (see paragraphs 4.5.4 and 4.9.3.11).

Project Synonymous with the term *database* in this guide except when referring to the PostgreSQL default database, postgres.

Prototype table A table created via the **Data | New table(s)** command, based on one of the table types. Instances of this table are created by using this table as the data type for a variable in a structure table. If this table is not referenced as a child in another table then it is also a root table.

Regular expression A special set of characters that defines a pattern for matching all or part of a text string. The search dialogs optionally allow the use of regular expressions in order to tailor a search more specifically (for example, the use of wild card characters). Information regarding the construction and use of regular expressions can be found on the web.

Root table The top-level table in the hierarchical table tree; the highest level ancestor of a child table. All non-structure tables and prototype tables not referenced as the data type for a variable in a structure table are root tables.

Structure type Data type that references a structure table prototype (the data type name is the structure prototype name).

Table path The path to a table beginning with its root table. For a non-structure table or other top-level table the table path is the root table name. For a child structure table the path lists the child's root table and all intervening ancestor tables and variable names in direct descent to the child table .

Table type A table template created using the Table Type Manager (see paragraph 4.9.3.104.9.3.7). Any number of tables may be created of a given table type.

Variable path The path to a variable beginning with its root table. For a non-structure table or other top-level table the variable path is the root table name. For a variable in a child structure table the path lists the child's root table and all intervening variable names in direct descent to the child table (identical to the table path, but without the structure names other than the root's).

Appendix C. Import and Export Format

Examples and descriptions of the CSV, EDS, JSON, and XTCE file formats used when importing and exporting tables are provided in the subsequent sections. For each format the identical project data is used to create the export output. This project data is shown in Figure 91 and Figure 92 (the data tables), Figure 93 and Figure 94 (the table type definitions), and Figure 95 (the data field, data type, and macro definitions).

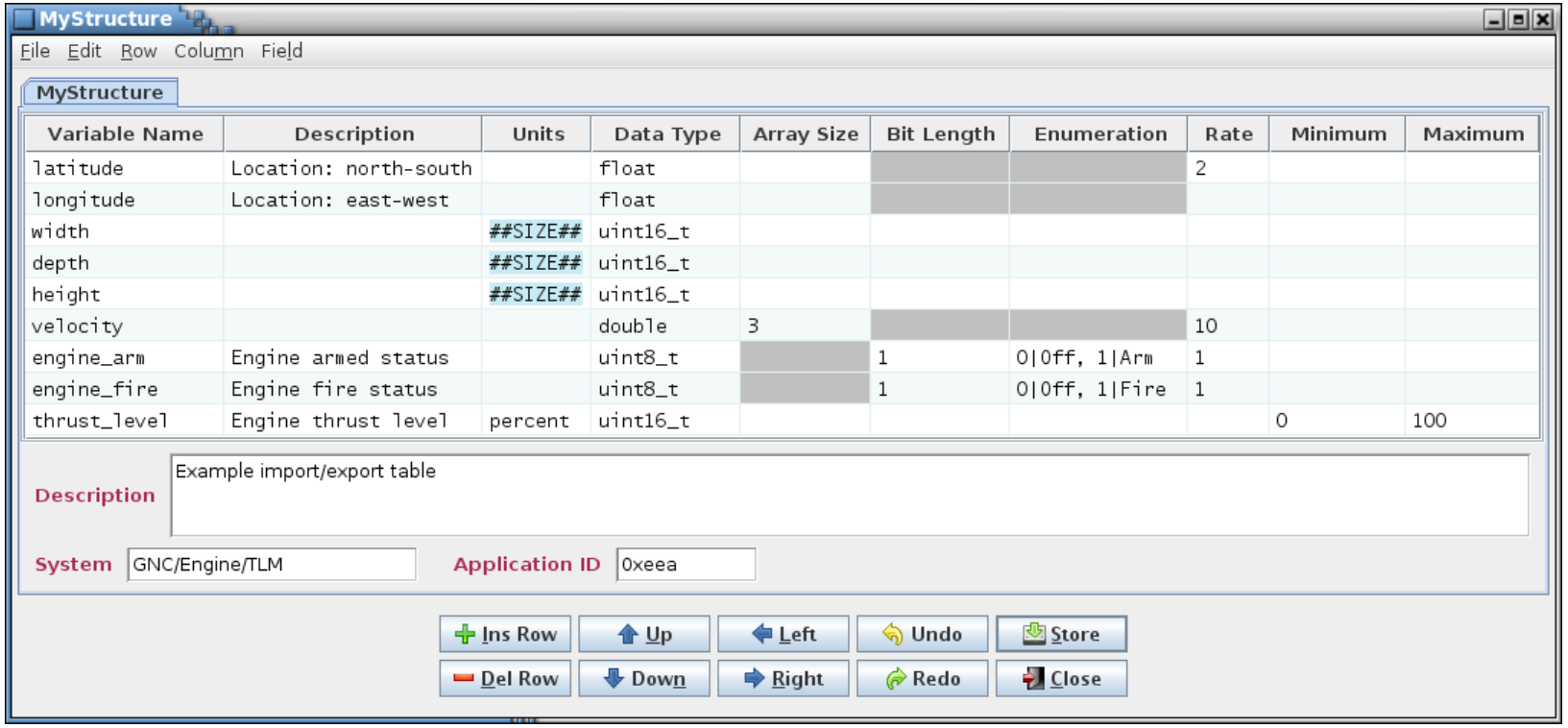


Figure 91. Structure table for import/export format examples

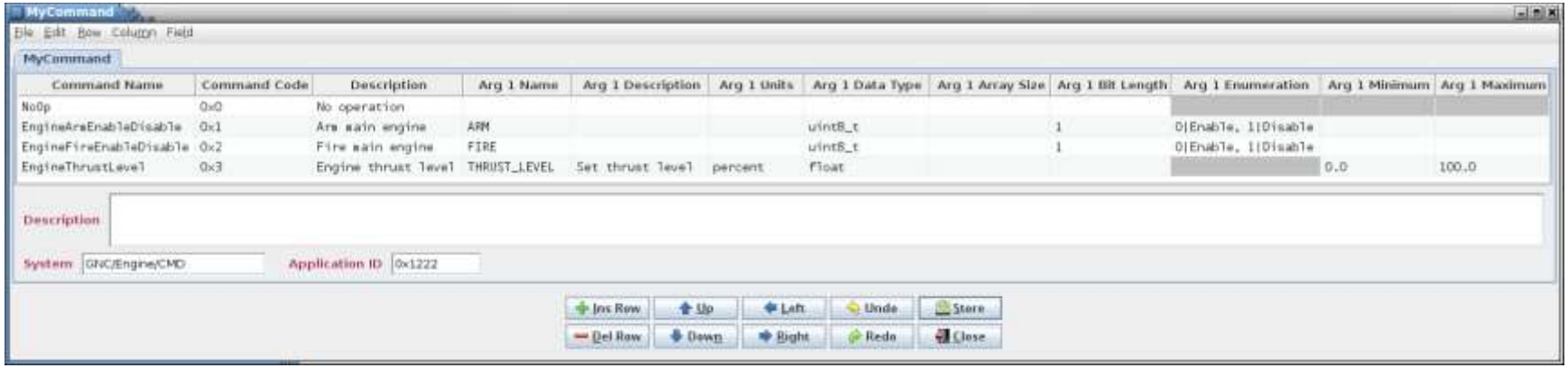


Figure 92. Command table for import/export examples

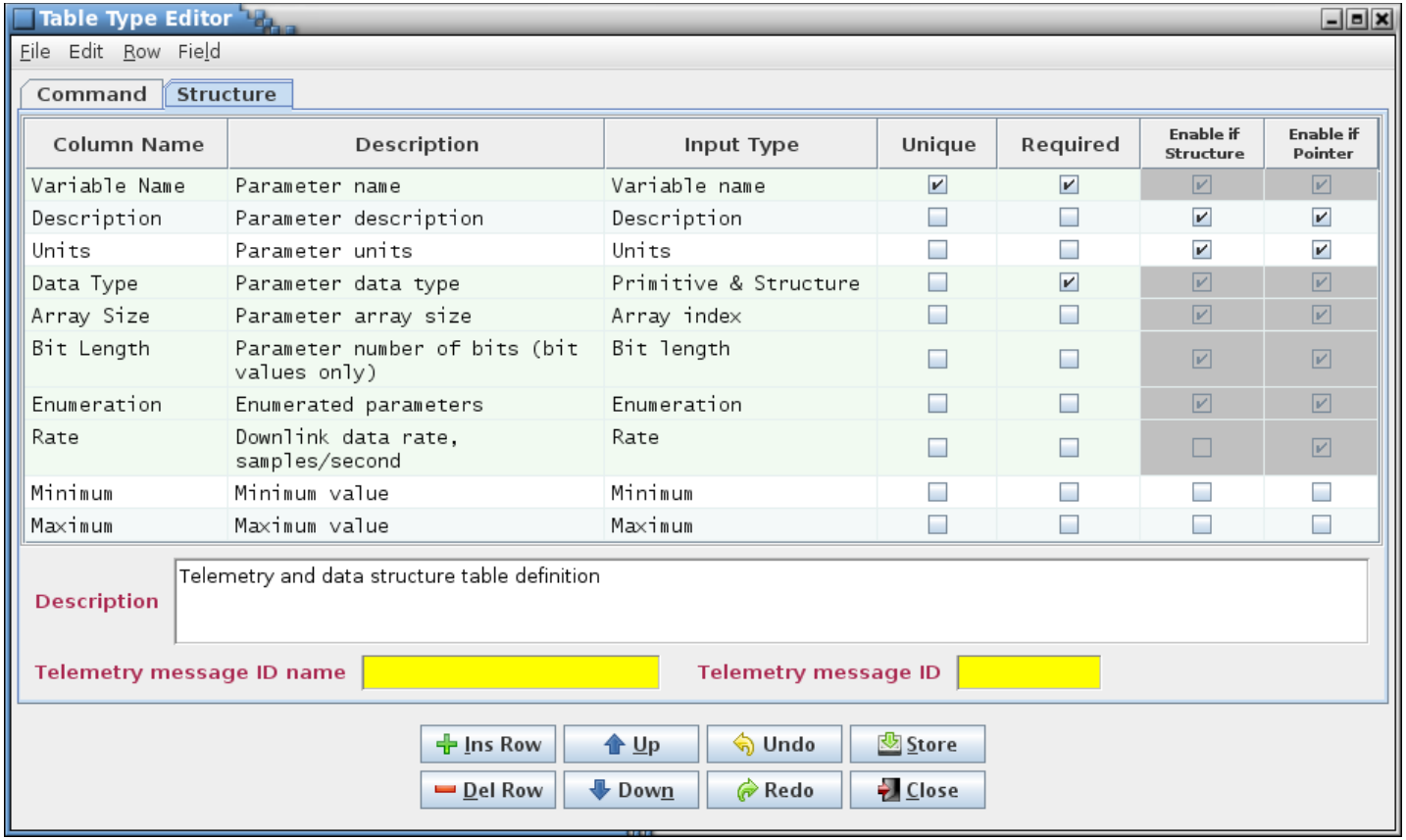


Figure 93. Structure table type definition for import/export example

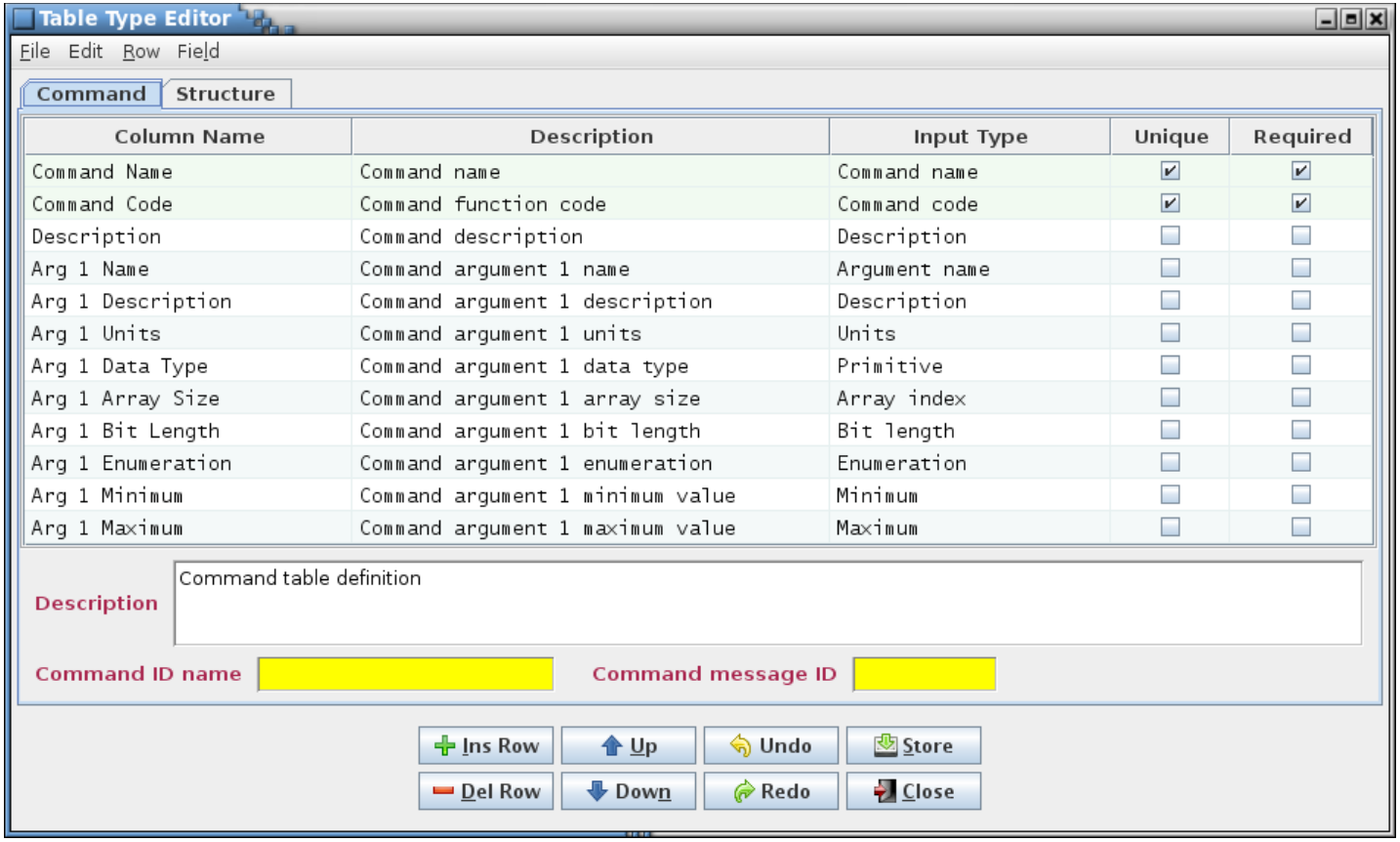


Figure 94. Command table type definition for import/export example

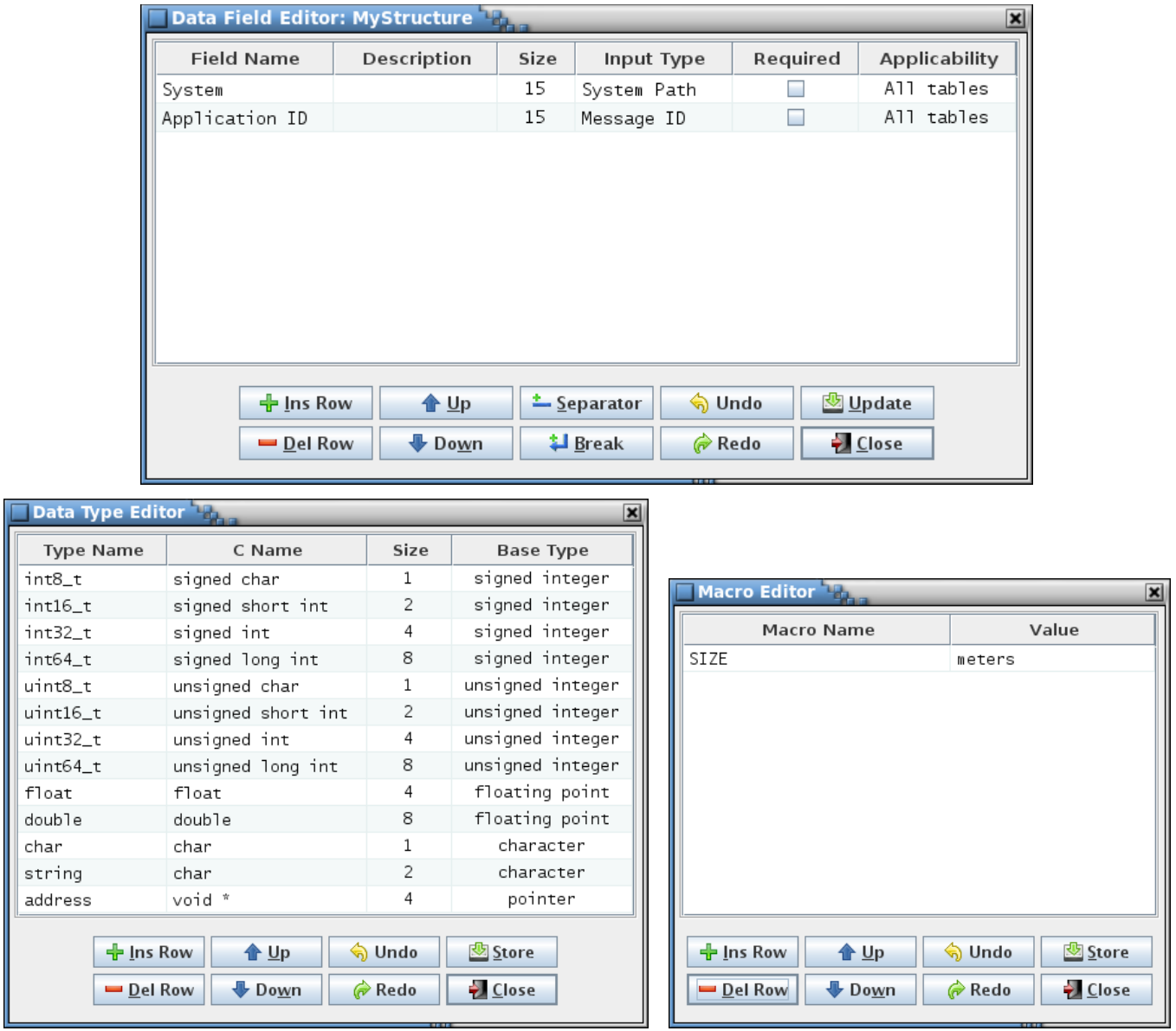


Figure 95. Data field, data type, and macro definitions for import/export example

Appendix C.1. CSV

The CSV import/export file is divided into five sections: table type definitions, data type definitions, macro definitions, reserved message IDs, and table definitions. These sections can appear in any order. Not all sections need be present. The table definitions are further sub-divided into table name & type, description, column data, and data field sections.

Each section is designated by a tag in the format *_tag_name_*. The subsequent row(s) are interpreted based on the last tag name until another tag is detected. The various values in the rows following a tag are separated by commas, with each value enclosed in double quotes in order to preserve quotes and commas in the values.

The name & type tag begins each table definition. The name & type and column data for at least one table must be present in the file.

Empty rows and rows beginning with a # character are ignored and can be used for section spacing and inserting comments. A '#' character beginning a quoted string is not interpreted as a comment (note that a spreadsheet application, if used to create the CSV file, may automatically surround text intended as a comment in quotes). Extra commas and white space characters appended to a row are ignored. This allows a CSV file generated from a spreadsheet application (which appends trailing commas to ensure that each row has the same number of columns) to be used without further editing.

The tag names and formats for the sections are as follows:

Table type definition section:

_table_type_

"*table type name*", "*table type description*"

"*type name*", "*type description*", "*input type*", "*unique*", "*required*", "*structure allowed*", "*pointer allowed*"

… repeat previous row for each table type and each type's column definitions

_data_fields_

"*field name*", "*description*", "*size in characters*", "*input type*", "*required*", "*applicability*", "*value*"

… repeat previous row for each data field associated with the table type. If left blank, default values for "size in characters" (10), "input type" (Text), and "applicability" (All tables) are used

See paragraph 4.9.3.10 for more information on the table type definition components.

Data type definition section:

_data_type_

"*data type name*", "*C type*", "*size in bytes*", "*base type*"

… *repeat previous row for each data type definition*

See paragraph 4.9.3.11 for more information on the data type definition components.

Macro definition section:

macros
"macro *name*", "*macro value*"
… *repeat previous row for each macro definition*

See paragraph 4.9.3.12 for more information on the macro definition components.

Reserved message ID section:

_ reserved_msg_ids_
"message ID (or ID range)", "message ID description"
… *repeat previous row for each reserved message ID (or ID range)*

See paragraph 4.9.3.14.2 for more information on the reserved message ID components.

Table definition section:

_name_type_
"*table path and name*", "*table type*"<, "*system name*">
description
"*table description*"
_column_data_
"*column 1 name*", "*column 2 name*", ….
"row 1 column 1 value", "row 1 column 2 value", …
"row 2 column 1 value", "row 2 column 2 value", …
… *repeat previous row for each row in the table*

_data_fields_

"field name", "description", "size in characters", "input type", "required", "applicability", "value"

… repeat previous row for each data field associated with the table. If left blank, default values for "size in characters" (10) and "input type" (Text) are used. The "applicability" input is unused by the table definition and may be blank

…repeat above, starting with _name_type_*, for each table definition*

The system name under _name_type_ is optional; when exporting this is the value of the data field with the **System path** input type (if present), but when importing the value is unused.

PROJECT-LEVEL FIELDS…

The example tables, MyStructure and MyCommand, are shown below as exported in CSV format.

```
# Created Wed Apr 11 14:29:44 CDT 2018 : project = user_s_guide : host = localhost:5432 : user = rmcclune

_name_type_
"MyCommand","Command","GNC/Engine/CMD"
_column_data_
"Command Name","Command Code","Description","Arg 1 Name","Arg 1 Description","Arg 1 Units","Arg 1 Data
Type","Arg 1 Array Size","Arg 1 Bit Length","Arg 1 Enumeration","Arg 1 Minimum","Arg 1 Maximum"
"NoOp","0x0","No operation","","","","","","","","","","",""
"EngineArmEnableDisable","0x1","Arm main engine","ARM","","","uint8_t","","1","0|Enable, 1|Disable","",""
"EngineFireEnableDisable","0x2","Fire main engine","FIRE","","","uint8_t","","1","0|Enable,
1|Disable","",""
"EngineThrustLevel","0x3","Engine thrust level","THRUST_LEVEL","Set thrust
level","percent","float","","","","","0.0","100.0"
_data_fields_
"System","","15","System Path","false","All tables","GNC/Engine/CMD"
"Application ID","","7","Message ID","false","All tables","0x1222"

_name_type_
"MyStructure","Structure","GNC/Engine/TLM"
_description_
"Example import/export table"
_column_data_
"Variable Name","Description","Units","Data Type","Array Size","Bit
Length","Enumeration","Rate","Minimum","Maximum"
"latitude","Location: north-south","","float","","","","2","",""
"longitude","Location: east-west","","float","","","","","",""
```

```
"width","","##SIZE##","uint16_t","","","","","",""
"depth","","##SIZE##","uint16_t","","","","","",""
"height","","##SIZE##","uint16_t","","","","","",""
"velocity","","","double","3","","","10","",""
"velocity[0]","X-direction","m/s","double","3","","","10","",""
"velocity[1]","Y-direction","m/s","double","3","","","10","",""
"velocity[2]","Z-direction","m/s","double","3","","","10","",""
"engine_arm","Engine armed status","","uint8_t","","1","0|Off, 1|Arm","1","",""
"engine_fire","Engine fire status","","uint8_t","","1","0|Off, 1|Fire","1","",""
"thrust_level","Engine thrust level","percent","uint16_t","","","","","0","100"
_data_fields_
"System","","15","System Path","false","All tables","GNC/Engine/TLM"
"Application ID","","7","Message ID","false","All tables","0xeea"

_table_type_
"Command","Command table definition"
"Command Name","Command name","Command name","true","true","false","true"
"Command Code","Command function code","Command code","true","true","false","true"
"Description","Command description","Description","false","false","false","true"
"Arg 1 Name","Command argument 1 name","Argument name","false","false","false","true"
"Arg 1 Description","Command argument 1 description","Description","false","false","false","true"
"Arg 1 Units","Command argument 1 units","Units","false","false","false","true"
"Arg 1 Data Type","Command argument 1 data type","Primitive","false","false","false","true"
"Arg 1 Array Size","Command argument 1 array size","Array index","false","false","false","true"
"Arg 1 Bit Length","Command argument 1 bit length","Bit length","false","false","false","true"
"Arg 1 Enumeration","Command argument 1 enumeration","Enumeration","false","false","false","true"
"Arg 1 Minimum","Command argument 1 minimum value","Minimum","false","false","false","true"
"Arg 1 Maximum","Command argument 1 maximum value","Maximum","false","false","false","true"
_table_type_data_fields_
"Command ID name","Command ID name","15","Message ID name","true","All tables",""
"Command message ID","Command message ID","7","Message ID","true","All tables",""

_table_type_
"Structure","Telemetry and data structure table definition"
"Variable Name","Parameter name","Variable name","true","true","true","true"
"Description","Parameter description","Description","false","false","true","true"
"Units","Parameter units","Units","false","false","true","true"
"Data Type","Parameter data type","Primitive & Structure","false","true","true","true"
"Array Size","Parameter array size","Array index","false","false","true","true"
"Bit Length","Parameter number of bits (bit values only)","Bit length","false","false","true","true"
"Enumeration","Enumerated parameters","Enumeration","false","false","true","true"
```

```
"Rate","Downlink data rate, samples/second","Rate","false","false","false","true"
"Minimum","Minimum value","Minimum","false","false","false","false"
"Maximum","Maximum value","Maximum","false","false","false","false"
_table_type_data_fields_
"Telemetry message ID name","Telemetry message ID name","15","Message ID name","true","Roots only",""
"Telemetry message ID","Telemetry message ID","7","Message ID","true","Roots only",""

_data_type_
"uint8_t","unsigned char","1","unsigned integer"
"uint16_t","unsigned short int","2","unsigned integer"
"float","float","4","floating point"
"double","double","8","floating point"

_macros_
"SIZE","meters"
```

Appendix C.2. EDS

The **Namespace** *name* field is contructed using the contents of the table's data field having a **System path** input type (if present) followed by the table's full path, with all invalid characters (spaces, commas, periods, etc.) converted to underscores (this is a constraint imposed on the *name* field by the EDS schema).

When exporting, the **Namespace** *shortDesciption* field is used to store the original table name with its full path. During importing this field is checked and if it exists and adheres to the constraints for a table path and name then it is used as the table name. If the field doesn't meet the requirements then the **Namespace** *name* field is used instead. The *name* field isn't used by default since it is not allowed to contain commas or periods, which are required for a table's full path.

Each **Namespace** defines a structure or command table. The **Interface** *name* field is set to "Telemetry" for structures, and "Command" for commands; however, the presence of a **ParameterSet** or **CommandSet** in the **Interface** determines how the information is assigned when the file is imported. Both a **ParameterSet** and a **CommandSet** can exist in the same **Namespace**, a condition that can exist if the EDS file is constructed by other means than via the CCDD export operation. Since the **Namespace** fields are used to set the table name, both a **ParameterSet** and a **CommandSet** within the **Namespace** would result in the creation of tables with duplicate names. This is avoided by appending "_tlm" to structure table name and "_cmd" to the command table name for this case.

The original integer data type size for parameters with a bit length is not preserved when exporting in EDS format. The **IntegerDataEncoding** *sizeInBits* field is used to set the bit length; however, there is no field for the overall size. When the file is imported the data type size for a bit-wise parameter is set to the smallest integer into which the number of bits will fit. This can lead to a difference in the original data type sizes which in turn can affect bit-packing.

The **Device Metadata** is used to store the telemetry header table name (if defined), the command header table name (if defined), the telemetry and command header table application ID variable name, and the command header command function code variable name. These are defined in project-level data fields (see Table 7). Default values are used for the application ID variable name ("applicationID") and the command function code variable name ("functionCode") if these fields are not defined in the project database.

The structure table identified by the project-level data field as the command header is treated specially during conversion. When exported, the structure is converted into a **CommandSet** and stored in the same manner a command table. The **Interface** *abstract* field is set to 'true'; this is used to indicate that the commands in this **Namespace** are common to the other command tables. When the file is imported the command header information is converted back into a structure.

When exporting, each root table (structure and command) is checked for the existence of data fields with the **Message ID** and **System path** input types. The contents of the field containing a system path is prepended to the table's path and the result is used to contruct the **Namespace** *name* field (after converting invalid characters to underscores as previously mentioned). When importing the file the data field for the system path is reconstructed from the **Namespace** *name* (the system path is separated from the table name portion using a '/' character; the system path may contain '/' characters as well). If the data field with a **Message ID** input type exists then its contents is used to set the application ID variable in the table's telemetry or command header in the export file (by setting the **ValueConstraint** *value* field in a **ContainerDataType** within the table's **DataTypeSet**). When imported, the data field is reconstructed to contain the ID value.

When exporting, the values in the command table's column with the input type **Command code** are stored in the export file by setting the **ValueConstraint** *value* field (in a **ContainerDataType** within the table's **DataTypeSet)** to the function code value. When the command table is imported, these values are placed in the table's command code column.

The table defined using the project-level data field as the command header is stored once in the export file and each command table **Namespace** references this single header definition. Since the telemetry header table is referenced in the root structure tables as a header variable's data type, each root structure table will have an individual instance of the header contructed in the export file. When importing, a single telemetry header structure table is recreated. The root structure telemetry header variable use this table as the data type reference.

The example tables, MyStructure and MyCommand, are shown below as exported in EDS format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DataSheet xmlns="http://www.ccsds.org/schema/sois/seds" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.ccsds.org/schema/sois http://www.ccsds.org/schema/sois/seds.xsd">
    <Device name="User_s_Guide">
        <LongDescription>Project used for the import and export example in Appendix C of the user's guide

Author: rmcclune
Generated by CCDD 1.2.11 (4-11-2018)
Date: Wed Apr 11 14:31:48 CDT 2018
Project: User's Guide
Host: localhost:5432
```

```
Endianess: big</LongDescription>
        <Metadata>
            <MetadataValueSet>
                <StringValue value="CCSDS_TLM_HDR" name="XML: Telemetry Header"/>
                <StringValue value="CCSDS_CMD_HDR" name="XML: Command Header"/>
                <StringValue value="applicationID" name="XML: Application ID"/>
                <StringValue value="functionCode" name="XML: Function Code"/>
            </MetadataValueSet>
        </Metadata>
    </Device>
    <Namespace name="GNC/Engine/CMD/MyCommand" shortDescription="MyCommand">
        <DataTypeSet>
            <EnumeratedDataType name="ARM_Type">
                <IntegerDataEncoding encoding="unsigned" sizeInBits="1" byteOrder="bigEndian"/>
                <EnumerationList>
                    <Enumeration value="0" label="Enable"/>
                    <Enumeration value="1" label="Disable"/>
                </EnumerationList>
            </EnumeratedDataType>
            <ContainerDataType baseType="CCSDS_CMD_HDR" name="EngineArmEnableDisable">
                <ConstraintSet>
                    <Constraint entry="applicationID">
                        <ValueConstraint value="0x1222"/>
                    </Constraint>
                    <Constraint entry="functionCode">
                        <ValueConstraint value="0x1"/>
                    </Constraint>
                </ConstraintSet>
            </ContainerDataType>
            <EnumeratedDataType name="FIRE_Type">
                <IntegerDataEncoding encoding="unsigned" sizeInBits="1" byteOrder="bigEndian"/>
                <EnumerationList>
                    <Enumeration value="0" label="Enable"/>
                    <Enumeration value="1" label="Disable"/>
                </EnumerationList>
            </EnumeratedDataType>
            <ContainerDataType baseType="CCSDS_CMD_HDR" name="EngineFireEnableDisable">
                <ConstraintSet>
                    <Constraint entry="applicationID">
                        <ValueConstraint value="0x1222"/>
                    </Constraint>
```

```
            <Constraint entry="functionCode">
                <ValueConstraint value="0x2"/>
            </Constraint>
        </ConstraintSet>
    </ContainerDataType>
    <FloatDataType name="THRUST_LEVEL_Type">
        <Semantics unit="percent"/>
        <FloatDataEncoding encodingAndPrecision="IEEE754_2008_single" byteOrder="bigEndian"/>
        <Range>
            <PrecisionRange>single</PrecisionRange>
        </Range>
    </FloatDataType>
    <ContainerDataType baseType="CCSDS_CMD_HDR" name="EngineThrustLevel">
        <ConstraintSet>
            <Constraint entry="applicationID">
                <ValueConstraint value="0x1222"/>
            </Constraint>
            <Constraint entry="functionCode">
                <ValueConstraint value="0x3"/>
            </Constraint>
        </ConstraintSet>
    </ContainerDataType>
</DataTypeSet>
<DeclaredInterfaceSet>
    <Interface name="Command">
        <BaseTypeSet>
            <BaseType type="CCSDS_CMD_HDR/Command"/>
        </BaseTypeSet>
        <CommandSet>
            <Command name="NoOp">
                <LongDescription>No operation</LongDescription>
            </Command>
            <Command name="EngineArmEnableDisable">
                <LongDescription>Arm main engine</LongDescription>
                <Argument type="ARM_Type" name="ARM"/>
            </Command>
            <Command name="EngineFireEnableDisable">
                <LongDescription>Fire main engine</LongDescription>
                <Argument type="FIRE_Type" name="FIRE"/>
            </Command>
            <Command name="EngineThrustLevel">
```

```
                    <LongDescription>Engine thrust level</LongDescription>
                    <Argument type="THRUST_LEVEL_Type" name="THRUST_LEVEL">
                        <LongDescription>Set thrust level</LongDescription>
                        <ValidRange/>
                    </Argument>
                </Command>
            </CommandSet>
        </Interface>
    </DeclaredInterfaceSet>
</Namespace>
<Namespace name="GNC/Engine/TLM/MyStructure" shortDescription="MyStructure">
    <LongDescription>Example import/export table</LongDescription>
    <DataTypeSet>
        <ContainerDataType name="MyStructure_Type">
            <EntryList>
                <Entry type="float_Type" name="latitude"/>
                <Entry type="float_Type" name="longitude"/>
                <Entry type="uint16_t_Type" name="width"/>
                <Entry type="uint16_t_Type" name="depth"/>
                <Entry type="uint16_t_Type" name="height"/>
                <Entry type="double_Array" name="velocity"/>
                <Entry type="uint8_t_Type" name="engine_arm"/>
                <Entry type="uint8_t_Type" name="engine_fire"/>
                <Entry type="uint16_t_Type" name="thrust_level">
                    <ValidRange>
                        <MinMaxRange min="0" max="100" rangeType="inclusiveMinInclusiveMax"/>
                    </ValidRange>
                </Entry>
            </EntryList>
        </ContainerDataType>
        <FloatDataType name="latitude_Type">
            <LongDescription>Location: north-south</LongDescription>
            <FloatDataEncoding encodingAndPrecision="IEEE754_2008_single" byteOrder="bigEndian"/>
            <Range>
                <PrecisionRange>single</PrecisionRange>
            </Range>
        </FloatDataType>
        <FloatDataType name="longitude_Type">
            <LongDescription>Location: east-west</LongDescription>
            <FloatDataEncoding encodingAndPrecision="IEEE754_2008_single" byteOrder="bigEndian"/>
            <Range>
```

```
        <PrecisionRange>single</PrecisionRange>
    </Range>
</FloatDataType>
<IntegerDataType name="width_Type">
    <IntegerDataEncoding encoding="unsigned" sizeInBits="16" byteOrder="bigEndian"/>
    <Range/>
</IntegerDataType>
<IntegerDataType name="depth_Type">
    <IntegerDataEncoding encoding="unsigned" sizeInBits="16" byteOrder="bigEndian"/>
    <Range/>
</IntegerDataType>
<IntegerDataType name="height_Type">
    <IntegerDataEncoding encoding="unsigned" sizeInBits="16" byteOrder="bigEndian"/>
    <Range/>
</IntegerDataType>
<ArrayDataType dataTypeRef="velocity_Type" name="velocity_Array">
    <DimensionList>
        <Dimension size="3"/>
    </DimensionList>
</ArrayDataType>
<FloatDataType name="velocity_Type">
    <FloatDataEncoding encodingAndPrecision="IEEE754_2008_double" byteOrder="bigEndian"/>
    <Range>
        <PrecisionRange>double</PrecisionRange>
    </Range>
</FloatDataType>
<EnumeratedDataType name="engine_arm_Type">
    <LongDescription>Engine armed status</LongDescription>
    <IntegerDataEncoding encoding="unsigned" sizeInBits="1" byteOrder="bigEndian"/>
    <EnumerationList>
        <Enumeration value="0" label="Off"/>
        <Enumeration value="1" label="Arm"/>
    </EnumerationList>
</EnumeratedDataType>
<EnumeratedDataType name="engine_fire_Type">
    <LongDescription>Engine fire status</LongDescription>
    <IntegerDataEncoding encoding="unsigned" sizeInBits="1" byteOrder="bigEndian"/>
    <EnumerationList>
        <Enumeration value="0" label="Off"/>
        <Enumeration value="1" label="Fire"/>
    </EnumerationList>
```

```
            </EnumeratedDataType>
            <IntegerDataType name="thrust_level_Type">
                <LongDescription>Engine thrust level</LongDescription>
                <Semantics unit="percent"/>
                <IntegerDataEncoding encoding="unsigned" sizeInBits="16" byteOrder="bigEndian"/>
                <Range/>
            </IntegerDataType>
        </DataTypeSet>
        <DeclaredInterfaceSet>
            <Interface name="Telemetry">
                <ParameterSet>
                    <Parameter type="latitude_Type" name="latitude">
                        <LongDescription>Location: north-south</LongDescription>
                    </Parameter>
                    <Parameter type="longitude_Type" name="longitude">
                        <LongDescription>Location: east-west</LongDescription>
                    </Parameter>
                    <Parameter type="width_Type" name="width"/>
                    <Parameter type="depth_Type" name="depth"/>
                    <Parameter type="height_Type" name="height"/>
                    <Parameter type="velocity_Array" name="velocity"/>
                    <Parameter type="engine_arm_Type" name="engine_arm">
                        <LongDescription>Engine armed status</LongDescription>
                    </Parameter>
                    <Parameter type="engine_fire_Type" name="engine_fire">
                        <LongDescription>Engine fire status</LongDescription>
                    </Parameter>
                    <Parameter type="thrust_level_Type" name="thrust_level">
                        <LongDescription>Engine thrust level</LongDescription>
                    </Parameter>
                </ParameterSet>
            </Interface>
        </DeclaredInterfaceSet>
    </Namespace>
</DataSheet>
```

Appendix C.3. JSON

A JSON object consists of a name/value pair where the value can be a single value or an array containing more name/value pairs. The CCDD JSON export file uses a unique object name for specific portions of the exported data. These object names are described below (these names are a subset of those used in the web server JSON output).

File Description	Information on the export file including the creation time and date, project name, database host, and user name.
Data Type Definition	An array containing an entry for each data type definition. Each definition has the following entries: Base Type, C Name, Type Name, and Size.
Table Type Definition	Array containing the table type definitions. Each array entry defines a table type and contains a **Table Type Column** array, **Table Type Name**, **Table Type Description**, and **Table Type Data Field** array.
Table Type Name	Table type definition name.
Table Type Description	Table type definition description.
Table Type Column	Array containing the table type column definitions. Each definition contains the Column Name, Description, Input Type, Required, Unique, Enable if Structure, and Enable if Pointer.
Table Type Data Field	Array of the table type's data field definitions. Each definition has the following entries: Value, Required, Description, Applicability, Field Name, Input Type, and Size.
Macro Definition	Array containing the macro definitions. Each definition contains the Macro Name and Macro Value.
Reserved Message ID Definition	Array containing the reserved message ID definitions. Each definition has entries for the Message ID(s) and Description.
Variable Path	Array containing the variable paths. Each entry contains the full variable path in the application's native format, and the path formatted per the export dialog options.
Table Definition	Table definition array. Each array entry in the table definition contains the **Table Data**, **Table Name**, **Table Description**, **Table Type**, **System Name**, and **Data Field**(s) for a single table.
Table Name	Table name, including its full path.
Table Type	Table type name.
Table Description	Table description.
Table Data	Array containing the table data. Each array entry consists of a column name and its corresponding value.
System Name	The value of the table's data field with the input type of **System path**, if present.
Data Field	Array of the table's data field definitions. Each definition has the following entries: Value, Required flag, Description, Applicability type, Field Name, Input Type, and Size.

Project Data Field

The example tables, MyStructure and MyCommand, are shown below as exported in JSON format.


```
{
  "Table Definition": [
    {
      "Table Data": [
        {
          "Command Name": "NoOp",
          "Description": "No operation",
          "Command Code": "0x0"
        },
        {
          "Command Name": "EngineArmEnableDisable",
          "Description": "Arm main engine",
          "Arg 1 Data Type": "uint8_t",
          "Command Code": "0x1",
          "Arg 1 Enumeration": "0|Enable, 1|Disable",
          "Arg 1 Bit Length": "1",
          "Arg 1 Name": "ARM"
        },
        {
          "Command Name": "EngineFireEnableDisable",
          "Description": "Fire main engine",
          "Arg 1 Data Type": "uint8_t",
          "Command Code": "0x2",
          "Arg 1 Enumeration": "0|Enable, 1|Disable",
          "Arg 1 Bit Length": "1",
          "Arg 1 Name": "FIRE"
        },
        {
          "Arg 1 Units": "percent",
          "Command Name": "EngineThrustLevel",
          "Description": "Engine thrust level",
          "Arg 1 Data Type": "float",
          "Arg 1 Description": "Set thrust level",
          "Arg 1 Maximum": "100.0",
          "Arg 1 Minimum": "0.0",
          "Command Code": "0x3",
          "Arg 1 Name": "THRUST_LEVEL"
        }
      ],
```

```
  "Table Name": "MyCommand",
  "System Name": "GNC/Engine/CMD",
  "Data Field": [
    {
      "Value": "GNC/Engine/CMD",
      "Required": false,
      "Description": "",
      "Applicability": "All tables",
      "Field Name": "System",
      "Input Type": "System Path",
      "Size": 15
    },
    {
      "Value": "0x1222",
      "Required": false,
      "Description": "",
      "Applicability": "All tables",
      "Field Name": "Application ID",
      "Input Type": "Message ID",
      "Size": 7
    }
  ],
  "Table Description": "",
  "Table Type": "Command"
},
{
  "Table Data": [
    {
      "Description": "Location: north-south",
      "Data Type": "float",
      "Rate": "2",
      "Variable Name": "latitude"
    },
    {
      "Description": "Location: east-west",
      "Data Type": "float",
      "Variable Name": "longitude"
    },
    {
      "Units": "meters",
      "Data Type": "uint16_t",
```

```
  "Variable Name": "width"
},
{
  "Units": "meters",
  "Data Type": "uint16_t",
  "Variable Name": "depth"
},
{
  "Units": "meters",
  "Data Type": "uint16_t",
  "Variable Name": "height"
},
{
  "Array Size": "3",
  "Data Type": "double",
  "Rate": "10",
  "Variable Name": "velocity"
},
{
  "Array Size": "3",
  "Description": "X-direction",
  "Units": "m/s",
  "Data Type": "double",
  "Rate": "10",
  "Variable Name": "velocity[0]"
},
{
  "Array Size": "3",
  "Description": "Y-direction",
  "Units": "m/s",
  "Data Type": "double",
  "Rate": "10",
  "Variable Name": "velocity[1]"
},
{
  "Array Size": "3",
  "Description": "Z-direction",
  "Units": "m/s",
  "Data Type": "double",
  "Rate": "10",
  "Variable Name": "velocity[2]"
```

```
  },
  {
    "Description": "Engine armed status",
    "Enumeration": "0|Off, 1|Arm",
    "Data Type": "uint8_t",
    "Bit Length": "1",
    "Rate": "1",
    "Variable Name": "engine_arm"
  },
  {
    "Description": "Engine fire status",
    "Enumeration": "0|Off, 1|Fire",
    "Data Type": "uint8_t",
    "Bit Length": "1",
    "Rate": "1",
    "Variable Name": "engine_fire"
  },
  {
    "Description": "Engine thrust level",
    "Units": "percent",
    "Maximum": "100",
    "Minimum": "0",
    "Data Type": "uint16_t",
    "Variable Name": "thrust_level"
  }
],
"Table Name": "MyStructure",
"System Name": "GNC/Engine/TLM",
"Data Field": [
  {
    "Value": "GNC/Engine/TLM",
    "Required": false,
    "Description": "",
    "Applicability": "All tables",
    "Field Name": "System",
    "Input Type": "System Path",
    "Size": 15
  },
  {
    "Value": "0xeea",
    "Required": false,
```

```
          "Description": "",
          "Applicability": "All tables",
          "Field Name": "Application ID",
          "Input Type": "Message ID",
          "Size": 7
        }
      ],
      "Table Description": "Example import/export table",
      "Table Type": "Structure"
    }
  ],
  "Data Type Definition": [
    {
      "Base Type": "unsigned integer",
      "C Name": "unsigned char",
      "Type Name": "uint8_t",
      "Size": "1"
    },
    {
      "Base Type": "floating point",
      "C Name": "float",
      "Type Name": "float",
      "Size": "4"
    },
    {
      "Base Type": "unsigned integer",
      "C Name": "unsigned short int",
      "Type Name": "uint16_t",
      "Size": "2"
    },
    {
      "Base Type": "floating point",
      "C Name": "double",
      "Type Name": "double",
      "Size": "8"
    }
  ],
  "File Description": "Created Wed Apr 11 14:32:46 CDT 2018 : project = user_s_guide : host =
localhost:5432 : user = rmcclune",
  "Table Type Definition": [
    {
```

```
"Table Type Column": [
  {
    "Required": true,
    "Description": "Command name",
    "Unique": true,
    "Column Name": "Command Name",
    "Enable if Structure": false,
    "Enable if Pointer": true,
    "Input Type": "Command name"
  },
  {
    "Required": true,
    "Description": "Command function code",
    "Unique": true,
    "Column Name": "Command Code",
    "Enable if Structure": false,
    "Enable if Pointer": true,
    "Input Type": "Command code"
  },
  {
    "Required": false,
    "Description": "Command description",
    "Unique": false,
    "Column Name": "Description",
    "Enable if Structure": false,
    "Enable if Pointer": true,
    "Input Type": "Description"
  },
  {
    "Required": false,
    "Description": "Command argument 1 name",
    "Unique": false,
    "Column Name": "Arg 1 Name",
    "Enable if Structure": false,
    "Enable if Pointer": true,
    "Input Type": "Argument name"
  },
  {
    "Required": false,
    "Description": "Command argument 1 description",
    "Unique": false,
```

```
  "Column Name": "Arg 1 Description",
  "Enable if Structure": false,
  "Enable if Pointer": true,
  "Input Type": "Description"
},
{
  "Required": false,
  "Description": "Command argument 1 units",
  "Unique": false,
  "Column Name": "Arg 1 Units",
  "Enable if Structure": false,
  "Enable if Pointer": true,
  "Input Type": "Units"
},
{
  "Required": false,
  "Description": "Command argument 1 data type",
  "Unique": false,
  "Column Name": "Arg 1 Data Type",
  "Enable if Structure": false,
  "Enable if Pointer": true,
  "Input Type": "Primitive"
},
{
  "Required": false,
  "Description": "Command argument 1 array size",
  "Unique": false,
  "Column Name": "Arg 1 Array Size",
  "Enable if Structure": false,
  "Enable if Pointer": true,
  "Input Type": "Array index"
},
{
  "Required": false,
  "Description": "Command argument 1 bit length",
  "Unique": false,
  "Column Name": "Arg 1 Bit Length",
  "Enable if Structure": false,
  "Enable if Pointer": true,
  "Input Type": "Bit length"
},
```

```
    {
      "Required": false,
      "Description": "Command argument 1 enumeration",
      "Unique": false,
      "Column Name": "Arg 1 Enumeration",
      "Enable if Structure": false,
      "Enable if Pointer": true,
      "Input Type": "Enumeration"
    },
    {
      "Required": false,
      "Description": "Command argument 1 minimum value",
      "Unique": false,
      "Column Name": "Arg 1 Minimum",
      "Enable if Structure": false,
      "Enable if Pointer": true,
      "Input Type": "Minimum"
    },
    {
      "Required": false,
      "Description": "Command argument 1 maximum value",
      "Unique": false,
      "Column Name": "Arg 1 Maximum",
      "Enable if Structure": false,
      "Enable if Pointer": true,
      "Input Type": "Maximum"
    }
  ],
  "Table Type Name": "Command",
  "Table Type Data Field": [
    {
      "Value": "",
      "Required": true,
      "Description": "Command ID name",
      "Applicability": "All tables",
      "Field Name": "Command ID name",
      "Input Type": "Message ID name",
      "Size": 15
    },
    {
      "Value": "",
```

```
      "Required": true,
      "Description": "Command message ID",
      "Applicability": "All tables",
      "Field Name": "Command message ID",
      "Input Type": "Message ID",
      "Size": 7
    }
  ],
  "Table Type Description": "Command table definition"
},
{
  "Table Type Column": [
    {
      "Required": true,
      "Description": "Parameter name",
      "Unique": true,
      "Column Name": "Variable Name",
      "Enable if Structure": true,
      "Enable if Pointer": true,
      "Input Type": "Variable name"
    },
    {
      "Required": false,
      "Description": "Parameter description",
      "Unique": false,
      "Column Name": "Description",
      "Enable if Structure": true,
      "Enable if Pointer": true,
      "Input Type": "Description"
    },
    {
      "Required": false,
      "Description": "Parameter units",
      "Unique": false,
      "Column Name": "Units",
      "Enable if Structure": true,
      "Enable if Pointer": true,
      "Input Type": "Units"
    },
    {
      "Required": true,
```

```
  "Description": "Parameter data type",
  "Unique": false,
  "Column Name": "Data Type",
  "Enable if Structure": true,
  "Enable if Pointer": true,
  "Input Type": "Primitive & Structure"
},
{
  "Required": false,
  "Description": "Parameter array size",
  "Unique": false,
  "Column Name": "Array Size",
  "Enable if Structure": true,
  "Enable if Pointer": true,
  "Input Type": "Array index"
},
{
  "Required": false,
  "Description": "Parameter number of bits (bit values only)",
  "Unique": false,
  "Column Name": "Bit Length",
  "Enable if Structure": true,
  "Enable if Pointer": true,
  "Input Type": "Bit length"
},
{
  "Required": false,
  "Description": "Enumerated parameters",
  "Unique": false,
  "Column Name": "Enumeration",
  "Enable if Structure": true,
  "Enable if Pointer": true,
  "Input Type": "Enumeration"
},
{
  "Required": false,
  "Description": "Downlink data rate, samples/second",
  "Unique": false,
  "Column Name": "Rate",
  "Enable if Structure": false,
  "Enable if Pointer": true,
```

```
    "Input Type": "Rate"
  },
  {
    "Required": false,
    "Description": "Minimum value",
    "Unique": false,
    "Column Name": "Minimum",
    "Enable if Structure": false,
    "Enable if Pointer": false,
    "Input Type": "Minimum"
  },
  {
    "Required": false,
    "Description": "Maximum value",
    "Unique": false,
    "Column Name": "Maximum",
    "Enable if Structure": false,
    "Enable if Pointer": false,
    "Input Type": "Maximum"
  }
],
"Table Type Name": "Structure",
"Table Type Data Field": [
  {
    "Value": "",
    "Required": true,
    "Description": "Telemetry message ID name",
    "Applicability": "Roots only",
    "Field Name": "Telemetry message ID name",
    "Input Type": "Message ID name",
    "Size": 15
  },
  {
    "Value": "",
    "Required": true,
    "Description": "Telemetry message ID",
    "Applicability": "Roots only",
    "Field Name": "Telemetry message ID",
    "Input Type": "Message ID",
    "Size": 7
  }
```

```
      ],
      "Table Type Description": "Telemetry and data structure table definition"
    }
  ]
}
```

Appendix C.4. XTCE

The **SpaceSystem** hierarchy is based on the contents of the table's data field having a **System path** input type (if present) and the table's full path. The system path is assumed to use the '/' character to define each level of the path. The table name becomes the final level in the hierarchy. For example, the table 'MyStructure' has the system path 'GNC/Power/TLM' and the table 'MyCommand' has the system path 'GNC/Power/CMD'. The SpaceSystem hierarchy in the export file would be:

```
<SpaceSystem name="GNC">
  <SpaceSystem name="Power">
    <SpaceSystem name="TLM">
      <SpaceSystem name="MyStructure">
      </SpaceSystem>
    </SpaceSystem>
    <SpaceSystem name="CMD">
      <SpaceSystem name="MyCommand">
      </SpaceSystem>
    </SpaceSystem>
  </SpaceSystem>
</SpaceSystem>
```

Each **SpaceSystem** with **TelemetryMetaData** or **CommandMetaData** defines a structure or command table, respectively. Both a **TelemetryMetaData** and a **CommandMetaData** can exist in the same **SpaceSystem**, a condition that can exist if the XTCE file is constructed by other means than via the CCDD export operation. Since the **SpaceSystem** *name* field is used to set the table name, both a **TelemetryMetaData** and a **CommandMetaData** within the same **SpaceSystem** would result in the creation of tables with duplicate names. This is avoided by appending "_tlm" to structure table name and "_cmd" to the command table name for this case.

The original integer data type size for parameters with a bit length is not preserved when exporting in XTCE format. The **IntegerDataEncoding** *sizeInBits* field is used to set the bit length; however, there is no field for the overall size. The **IntegerDataType** also has a *sizeInBits* field which could be used to store the size of the integer; however, the **EnumeratedDataType** lacks this field. Rather than have one case covered (integers) and the other not (enumerations), and since the convention would be arbitrary (i.e., which *sizeInBits* field contains the integer size and which is the bit length), both are treated the same: when the file is imported the data type size for a bit-wise parameter is set to the smallest integer into which the number of bits will fit. This can lead to a difference in the original data type sizes which in turn can affect bit-packing.

The root space system **AncillaryData** is used to store the telemetry header table name (if defined), the command heard table name (if defined), the telemetry and command header table application ID variable name, and the command header command function code variable name. These are defined in project-level data fields (see Table 7). Default values are used for the application ID variable name ("applicationID") and the command function code variable name ("functionCode") if these fields are not defined in the project database.

The structure table identified by the project-level data field as the command header is treated specially during conversion. When exported, the structure is converted into **CommandMetaData** and stored in the same manner a command table. The **MetaCommand** *abstract* field is set to 'true'; this is used to indicate that the commands in this **SpaceSystem** are common to the other command tables. When the file is imported the command header information is converted back into a structure.

When exporting, each root table (structure and command) is checked for the existence of data fields with the **Message ID** and **System path** input types. The contents of the field containing a system path is used to establish the SpaceSystem hierarchy as described previously. When importing the file the data field for the system path is reconstructed from the **SpaceSystem** *name* fields preceding it in the table's hierarchy (each **SpaceSystem** *name* is separated by a '/' character). If the data field with a **Message ID** input type exists then its contents is used to set the application ID variable in the table's telemetry or command header in the export file (by setting the **Comparison** *value* field as a **RestrictionCriteria** within a **SequenceContainer** in a **ContainerSet** within the table's **TelemetryMetaData** for structure tables, and by setting the **ArgumentAssignment** *argumentValue* field within the **BaseMetaCommand** in the **CommandMetaData** for command tables). When imported, the data field is reconstructed to contain the ID value.

When exporting, the values in the command table's column with the input type **Command code** are stored in the export file by setting the **ArgumentAssignment** *argumentValue* field within the **BaseMetaCommand** in the **CommandMetaData)** to the function code value. When the command table is imported, these values are placed in the table's command code column.

The table defined using the project-level data field as the command header is stored once in the export file and each command table **SpaceSystem** references this single header definition. Since the telemetry header table is referenced in the root structure tables as a header variable's data type, each root structure table will have an individual instance of the header contructed in the export file. When importing, a single telemetry header structure table is recreated. The root structure telemetry header variable use this table as the data type reference.

The example tables, MyStructure and MyCommand, are shown below as exported in XTCE format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpaceSystem xmlns="http://www.omg.org/space/xtce" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
name="User's Guide" xsi:schemaLocation="https://www.omg.org/spec/XTCE/20061101
https://www.omg.org/spec/XTCE/20061101/06-11-06.xsd">
    <LongDescription>Project used for the import and export example in Appendix C of the user's
guide</LongDescription>
    <AncillaryDataSet>
        <AncillaryData name="XML: Telemetry Header">CCSDS_TLM_HDR</AncillaryData>
        <AncillaryData name="XML: Command Header">CCSDS_CMD_HDR</AncillaryData>
```

```
        <AncillaryData name="XML: Application ID">applicationID</AncillaryData>
        <AncillaryData name="XML: Function Code">functionCode</AncillaryData>
    </AncillaryDataSet>
    <Header version="1.0" date="Wed Apr 11 14:32:59 CDT 2018" classification="DOMAIN"
validationStatus="Working">
        <AuthorSet>
            <Author>rmcclune</Author>
        </AuthorSet>
        <NoteSet>
            <Note>Generated by CCDD 1.2.11 (4-11-2018)</Note>
            <Note>Date: Wed Apr 11 14:32:59 CDT 2018</Note>
            <Note>Project: User's Guide</Note>
            <Note>Host: localhost:5432</Note>
            <Note>Endianess: big</Note>
        </NoteSet>
    </Header>
    <SpaceSystem name="GNC">
        <Header version="1.0" classification="SYSTEM" validationStatus="Working"/>
        <SpaceSystem name="Engine">
            <Header version="1.0" classification="SYSTEM" validationStatus="Working"/>
            <SpaceSystem name="CMD">
                <Header version="1.0" classification="SYSTEM" validationStatus="Working"/>
                <SpaceSystem name="MyCommand">
                    <Header version="1.0" classification="INTERFACE" validationStatus="Working"/>
                    <CommandMetaData>
                        <ArgumentTypeSet>
                            <EnumeratedArgumentType name="ARM_Type">
                                <IntegerDataEncoding encoding="unsigned" sizeInBits="1"
bitOrder="mostSignificantBitFirst"/>
                                <EnumerationList>
                                    <Enumeration value="0" label="Enable"/>
                                    <Enumeration value="1" label="Disable"/>
                                </EnumerationList>
                            </EnumeratedArgumentType>
                            <EnumeratedArgumentType name="FIRE_Type">
                                <IntegerDataEncoding encoding="unsigned" sizeInBits="1"
bitOrder="mostSignificantBitFirst"/>
                                <EnumerationList>
                                    <Enumeration value="0" label="Enable"/>
                                    <Enumeration value="1" label="Disable"/>
                                </EnumerationList>
```

```
                        </EnumeratedArgumentType>
                        <FloatArgumentType sizeInBits="32" name="THRUST_LEVEL_Type">
                            <LongDescription>Set thrust level</LongDescription>
                            <FloatDataEncoding encoding="IEEE754_1985" sizeInBits="32"/>
                            <ValidRangeSet>
                                <ValidRange minExclusive="0.0" maxExclusive="100.0"/>
                            </ValidRangeSet>
                        </FloatArgumentType>
                    </ArgumentTypeSet>
                    <MetaCommandSet>
                        <MetaCommand name="NoOp">
                            <LongDescription>No operation</LongDescription>
                        </MetaCommand>
                        <MetaCommand name="EngineArmEnableDisable">
                            <LongDescription>Arm main engine</LongDescription>
                            <BaseMetaCommand metaCommandRef="/User's
Guide/CCSDS_CMD_HDR/CCSDS_CMD_HDR">
                                <ArgumentAssignmentList>
                                    <ArgumentAssignment argumentName="applicationID"
argumentValue="0x1222"/>
                                    <ArgumentAssignment argumentName="functionCode"
argumentValue="0x1"/>
                                </ArgumentAssignmentList>
                            </BaseMetaCommand>
                            <ArgumentList>
                                <Argument argumentTypeRef="ARM_Type" name="ARM"/>
                            </ArgumentList>
                            <CommandContainer>
                                <EntryList>
                                    <ArgumentRefEntry argumentRef="ARM"/>
                                </EntryList>
                            </CommandContainer>
                        </MetaCommand>
                        <MetaCommand name="EngineFireEnableDisable">
                            <LongDescription>Fire main engine</LongDescription>
                            <BaseMetaCommand metaCommandRef="/User's
Guide/CCSDS_CMD_HDR/CCSDS_CMD_HDR">
                                <ArgumentAssignmentList>
                                    <ArgumentAssignment argumentName="applicationID"
argumentValue="0x1222"/>
```

```
                                    <ArgumentAssignment argumentName="functionCode"
argumentValue="0x2"/>
                                </ArgumentAssignmentList>
                            </BaseMetaCommand>
                            <ArgumentList>
                                <Argument argumentTypeRef="FIRE_Type" name="FIRE"/>
                            </ArgumentList>
                            <CommandContainer>
                                <EntryList>
                                    <ArgumentRefEntry argumentRef="FIRE"/>
                                </EntryList>
                            </CommandContainer>
                        </MetaCommand>
                        <MetaCommand name="EngineThrustLevel">
                            <LongDescription>Engine thrust level</LongDescription>
                            <BaseMetaCommand metaCommandRef="/User's
Guide/CCSDS_CMD_HDR/CCSDS_CMD_HDR">
                                <ArgumentAssignmentList>
                                    <ArgumentAssignment argumentName="applicationID"
argumentValue="0x1222"/>
                                    <ArgumentAssignment argumentName="functionCode"
argumentValue="0x3"/>
                                </ArgumentAssignmentList>
                            </BaseMetaCommand>
                            <ArgumentList>
                                <Argument argumentTypeRef="THRUST_LEVEL_Type" name="THRUST_LEVEL"/>
                            </ArgumentList>
                            <CommandContainer>
                                <EntryList>
                                    <ArgumentRefEntry argumentRef="THRUST_LEVEL"/>
                                </EntryList>
                            </CommandContainer>
                        </MetaCommand>
                    </MetaCommandSet>
                </CommandMetaData>
            </SpaceSystem>
        </SpaceSystem>
        <SpaceSystem name="TLM">
            <Header version="1.0" classification="SYSTEM" validationStatus="Working"/>
            <SpaceSystem name="MyStructure">
                <LongDescription>Example import/export table</LongDescription>
```

```
        <Header version="1.0" classification="INTERFACE" validationStatus="Working"/>
        <TelemetryMetaData>
            <ParameterTypeSet>
                <FloatParameterType sizeInBits="32" name="latitude_Type">
                    <LongDescription>Location: north-south</LongDescription>
                    <FloatDataEncoding encoding="IEEE754_1985" sizeInBits="32"/>
                </FloatParameterType>
                <FloatParameterType sizeInBits="32" name="longitude_Type">
                    <LongDescription>Location: east-west</LongDescription>
                    <FloatDataEncoding encoding="IEEE754_1985" sizeInBits="32"/>
                </FloatParameterType>
                <IntegerParameterType sizeInBits="16" signed="false" name="width_Type">
                    <UnitSet>
                        <Unit>meters</Unit>
                    </UnitSet>
                    <IntegerDataEncoding encoding="unsigned" sizeInBits="16"
bitOrder="mostSignificantBitFirst"/>
                </IntegerParameterType>
                <IntegerParameterType sizeInBits="16" signed="false" name="depth_Type">
                    <UnitSet>
                        <Unit>meters</Unit>
                    </UnitSet>
                    <IntegerDataEncoding encoding="unsigned" sizeInBits="16"
bitOrder="mostSignificantBitFirst"/>
                </IntegerParameterType>
                <IntegerParameterType sizeInBits="16" signed="false" name="height_Type">
                    <UnitSet>
                        <Unit>meters</Unit>
                    </UnitSet>
                    <IntegerDataEncoding encoding="unsigned" sizeInBits="16"
bitOrder="mostSignificantBitFirst"/>
                </IntegerParameterType>
                <ArrayParameterType arrayTypeRef="velocity_Type" numberOfDimensions="1"
name="velocity_Array"/>
                <FloatParameterType sizeInBits="64" name="velocity_Type">
                    <FloatDataEncoding encoding="IEEE754_1985" sizeInBits="64"/>
                </FloatParameterType>
                <EnumeratedParameterType name="engine_arm_Type">
                    <LongDescription>Engine armed status</LongDescription>
                    <IntegerDataEncoding encoding="unsigned" sizeInBits="1"
bitOrder="mostSignificantBitFirst"/>
```

```
                                <EnumerationList>
                                    <Enumeration value="0" label="Off"/>
                                    <Enumeration value="1" label="Arm"/>
                                </EnumerationList>
                            </EnumeratedParameterType>
                            <EnumeratedParameterType name="engine_fire_Type">
                                <LongDescription>Engine fire status</LongDescription>
                                <IntegerDataEncoding encoding="unsigned" sizeInBits="1"
bitOrder="mostSignificantBitFirst"/>
                                <EnumerationList>
                                    <Enumeration value="0" label="Off"/>
                                    <Enumeration value="1" label="Fire"/>
                                </EnumerationList>
                            </EnumeratedParameterType>
                            <IntegerParameterType sizeInBits="16" signed="false" name="thrust_level_Type">
                                <LongDescription>Engine thrust level</LongDescription>
                                <UnitSet>
                                    <Unit>percent</Unit>
                                </UnitSet>
                                <IntegerDataEncoding encoding="unsigned" sizeInBits="16"
bitOrder="mostSignificantBitFirst"/>
                                <ValidRange minInclusive="0" maxInclusive="100"/>
                            </IntegerParameterType>
                        </ParameterTypeSet>
                        <ParameterSet>
                            <Parameter parameterTypeRef="latitude_Type" name="latitude"/>
                            <Parameter parameterTypeRef="longitude_Type" name="longitude"/>
                            <Parameter parameterTypeRef="width_Type" name="width"/>
                            <Parameter parameterTypeRef="depth_Type" name="depth"/>
                            <Parameter parameterTypeRef="height_Type" name="height"/>
                            <Parameter parameterTypeRef="velocity_Array" name="velocity"/>
                            <Parameter parameterTypeRef="engine_arm_Type" name="engine_arm"/>
                            <Parameter parameterTypeRef="engine_fire_Type" name="engine_fire"/>
                            <Parameter parameterTypeRef="thrust_level_Type" name="thrust_level"/>
                        </ParameterSet>
                        <ContainerSet>
                            <SequenceContainer name="MyStructure">
                                <EntryList>
                                    <ParameterRefEntry parameterRef="latitude"/>
                                    <ParameterRefEntry parameterRef="longitude"/>
                                    <ParameterRefEntry parameterRef="width"/>
```

```
                        <ParameterRefEntry parameterRef="depth"/>
                        <ParameterRefEntry parameterRef="height"/>
                        <ArrayParameterRefEntry parameterRef="velocity">
                            <DimensionList>
                                <Dimension>
                                    <StartingIndex>
                                        <FixedValue>0</FixedValue>
                                    </StartingIndex>
                                    <EndingIndex>
                                        <FixedValue>3</FixedValue>
                                    </EndingIndex>
                                </Dimension>
                            </DimensionList>
                        </ArrayParameterRefEntry>
                        <ParameterRefEntry parameterRef="engine_arm"/>
                        <ParameterRefEntry parameterRef="engine_fire"/>
                        <ParameterRefEntry parameterRef="thrust_level"/>
                    </EntryList>
                </SequenceContainer>
            </ContainerSet>
        </TelemetryMetaData>
    </SpaceSystem>
</SpaceSystem>
</SpaceSystem>
</SpaceSystem>
```

Appendix D. Error & Warning Messages

The table below lists all of the error and warning messages, in alphabetical order, that can occur in the CCDD application and the causes. An error message implies that the intended operation cannot be successfully completed. An attempt is automatically made to revert any changes made to the project database in the event an error occurs during a database update. If this reversion is unsuccessful then the database is likely corrupted. A command line error message results in immediate program termination, but for other errors the application continues to run. A warning message indicates that although the operation was unsuccessful the user can effect a change to correct the problem.

Type	Message	Cause
Warning	# array member row(s) ignored due to missing array definition(s)	The number of rows indicated, #, were ignored when pasting data into a table. The cause is that one or more rows in the pasted data represent an array member, but an array definition does not precede the member(s). Include the array definition row when pasting array member information
Warning	All application parameters must be entered	An input text field is empty in the application parameter dialog. Enter a valid value in each of the fields
Warning	An association with this script and table(s) already exists in the script associations table	A script association using the same script file and data table(s) is already present in the script association table in the script manager
Error	An unanticipated error occurred; cause '*error cause*'. Error trace: *class name*: *method name*() line *line number*, <*further trace*>	An exception occurred that is not otherwise covered by the error handling routines. The cause is specified by *error cause*, followed by a method trace showing the line where the error occurred and the calls sequence leading to it
Warning	Application parameter values must be positive integer values	The value in one or more application parameter dialog input text fields contains a zero, negative, or non-integer value. Enter an integer value greater than or equal to 1 in each of the fields
Warning	Association name already in use	The association name entered in the script association manager is already in use by another association. Association names must be unique. Alter the association name to one not in use
Warning	At least one data stream must be selected	No target data stream is selected in the link copy dialog when the Okay button is pressed. Choose at least one data stream or press the Cancel button
Warning	Auto-fill detected mismatched rates for variable(s) associated with *variable path+name*	The telemetry scheduler auto-fill operation determined that one or more variables that are associated (via bit-packing or string membership) with the variable *variable path+name* do not have the same rate. Open the table containing the specified variable and adjust the rate for the associated variable(s)

Type	Message	Cause
Warning	Auto-fill unable to assign *number* applications	The application scheduler was unable to assign *number* applications to a time slot
Warning	Auto-fill unable to assign *number* variables	The telemetry scheduler was unable to assign *number* variables to an output message
Warning	Base data type inconsistent with data type usage in table(s) '*table name(s)*'	The base data type entered in the data type editor's Base Type column was changed from an integer (signed or unsigned) to a non-integer, and the indicated table(s), *table name(s)*, has a non-empty bit length or enumeration column. The associated data type for a bit length parameter or an enumerated parameter must be an integer. Clear the bit length and enumeration columns for the table(s) referencing this data type and then change the base type
Warning	Bit length exceeds the size of the data type in table(s) '*table name(s)*'	The size entered in the data type editor's Base Type column for an integer base type (signed or unsigned) was reduced and the data type is used with parameters having a bit length specified that exceeds the capacity of the new size. Reduce or clear the bit length for the table(s) referencing this data type and then change the size
Warning	Cannot assign application to a time slot	The application scheduler was unable to assign an application to a time slot when the user attempted manual assignment
Warning	Cannot assign variable to a message	The telemetry scheduler was unable to assign a variable to an output message when the user attempted manual assignment
Warning	Cannot close backup file '*backup file name*'	An error occurred preventing closing the backup file or temporary backup file *backup file name* when restoring a backup file
Warning	Cannot close export file '*path+file name*'	The export file failed to close after being written
Warning	Cannot close import file '*path+file name*'	The import file failed to close after being read
Error	Cannot close project database '*database name*'	An error occurred preventing closing project database *database name*. Detail on the cause is logged in the event log
Warning	Cannot close script file '*path+file name*'	The script file failed to close after being read
Error	Cannot close server connection	An error occurred preventing closing the PostgreSQL server connection (which is accomplished by closing the connection to the default database, postgres). Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot connect to project database '*database name*'	An attempt to connect to the project database *database name* failed. Detail on the cause is logged in the event log. This can occur due to lack of access permission by the user to the database, if the selected project is already open by another instance of the CCDD application, or if the locked status flag remained set due to abnormal application termination
Error	Cannot connect to server	An attempt to connect to the PostgreSQL server, (accomplished by connecting to the default database, postgres) failed. Detail on the cause is logged in the event log. This may occur if the postgreSQL server is not running
Error	Cannot copy project '*project name*'	An error occurred preventing copying of the project *project name*. Detail on the cause is logged in the event log
Error	Cannot copy table '*table name*'	The attempt to copy table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot copy table type '*table type*'	The attempt to copy table type *table type* in the project database failed. Detail on the cause is logged in the event log
Warning	Cannot create event log file	The event log file cannot be created. Check that file permissions allow read/write operations to the directory in which the CCDD application was executed
Error	Cannot create export file '*path+file name*'	The export .csv file *file name* cannot be created in the directory *path*. Check that the file permissions allow the user to write to this directory
Error	Cannot create output file '*path+file name*'	The output file *file name* cannot be created in the directory *path*. Check that the file permissions allow the user to write to this directory
Error	Cannot create project '*project name*'	An error occurred preventing creation of the project *project name*. Detail on the cause is logged in the event log
Error	Cannot create script file '*path+file name*'	The script file *file name* cannot be created in the directory *path*. Check that the file permissions allow the user to write to this directory
Error	Cannot create structure functions in project database '*database name*'	The SQL and pgplsql functions related to structure tables cannot be created in the project database *database name*. Detail on the cause is logged in the event log. This can occur due to lack of access permission by the user to the database
Error	Cannot create tables '*table names*'	The attempt to create tables *table names* in the project database failed. Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot create tables and functions in project database '*database name*'	The SQL and pgplsql functions and/or the default tables cannot be created in the project database *database name*. Detail on the cause is logged in the event log. This can occur due to lack of access permission by the user to the database
Error	Cannot create web server	The attempt to instantiate the embedded Jetty web server failed. Detail on the cause is logged in the event log
Warning	Cannot delete data type '*data type*'; data type is referenced by table(s) '*table name(s)*'	An attempt was made to delete the data type *data type*, but the data type is in use in the data table(s) *table name(s)*. A data type can't be removed until all references to it are first eliminated. Remove the data type reference(s) and then delete the data type
Warning	Cannot delete input type '*input type*'; input type is referenced by table(s) '*table name(s)*'	An attempt was made to delete the input type *input type*, but the input type is in use in the data table(s) *table name(s)*. An input type can't be removed until all references to it are first eliminated. Remove the input type reference(s) and then delete the input type
Warning	Cannot delete macro '*macro name*'; macro is referenced by table(s) '*table name(s)*'	An attempt was made to delete the macro *macro name*, but the macro is in use in the data table(s) *table name(s)*. A macro can't be removed until all references to it are first eliminated. Remove the macro reference(s) and then delete the macro
Error	Cannot delete project '*project name*'	An error occurred preventing deletion of the project *project name*. Detail on the cause is logged in the event log
Error	Cannot delete table type '*table type*' <and table(s) '*table name(s)*'>	The attempt to delete table type *table type* and its associated table(s) *table name(s)*, if any, from the project database failed. Detail on the cause is logged in the event log
Error	Cannot delete table(s) '*table name(s)*'	The attempt to delete table(s) *table name(s)* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot disable auto-commit	The attempt to disable the auto-commit mode for database changes failed. If this occurs subsequent database transactions are likely to fail. Restart the application; the affected project database may require manual unlocking. Detail on the cause is logged in the event log
Error	Cannot execute script '*script name*' using table(s) '*table name(s)*'	An error occurred during execution preparation of the script *script name*. Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot export to file '*file name*': Error in script function '*function name*'; cause '*error cause*'	Exporting the Table(s) to file *file name* in XTCE XML format failed due to an error in the external script file function *function name*, used to override the internal method of the same name. Information on the cause is displayed
Error	Cannot export to file '*file name*': *IO exception*	Exporting the Table(s) to file *file name* failed due to the I/O exception *IO exception* (the file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason)
Error	Cannot export to file '*file name*': *JAXB or Marshal exception*	Exporting the Table(s) to file *file name* in EDS or XTCE XML format failed due to the JAXB or Marshal exception *JAXB or Marshal exception*.
Error	Cannot export to file '*file name*': *Script exception*	Exporting the Table(s) to file *file name* in JSON format failed due to the script exception *script exception* (a JavaScript engine executes a script command that parses the output in JSON format for the export file)
Error	Cannot import EDS XML from file '*file name*'; cause '*error cause'*	Importing the table(s) from file *file name* in EDS XML format failed due to the specified cause
Error	Cannot import from file '*file name*' into table: Cannot locate file	Importing the data from file *file name* failed due to the file not existing. Check the file path and name
Error	Cannot import from file '*file name*' into table: EDS conversion setup failed; cause '*error cause*'	Importing the data from file *file name* failed during creation of the EDS XML marshaller/unmarshaller due to the specified cause
Error	Cannot import from file '*file name*' into table: File format invalid	The selected import CSV-formatted file *file name* is not in the expected format. Correct the import file format or select another file to import
Error	Cannot import from file '*file name*' into table: Imported data type '*data type name*' doesn't match the existing definition	Importing the data from file *file name* failed due to the data type '*data type name*' already existing in the project, but with a different definition than that in the import file. Delete the data type in the project or import file, or adjust the types to match
Error	Cannot import from file '*file name*' into table: Imported input type '*input type name*' doesn't match the existing definition	Importing the data from file *file name* failed due to the input type '*input type name*' already existing in the project, but with a different definition than that in the import file. Delete the input type in the project or import file, or adjust the types to match
Error	Cannot import from file '*file name*' into table: Imported macro '*macro name*' doesn't match the existing definition	Importing the data from file *file name* failed due to the macro '*macro name*' already existing in the project, but with a different definition than that in the import file. Delete the macro in the project or import file, or adjust the macros to match

Type	Message	Cause
Error	Cannot import from file '*file name*' into table: Imported project data field '*field name*' doesn't match the existing definition	Importing the data from file *file name* failed due to the project data field '*field name*' already existing in the project, but with a different definition than that in the import file. Delete the data field in the project or import file, or adjust the fields to match
Error	Cannot import from file '*file name*' into table: Imported table type '*table type name*' doesn't match the existing definition	Importing the data from file *file name* failed due to the table type '*table type name*' already existing in the project, but with a different definition than that in the import file. Delete the table type in the project or import file, or adjust the types to match
Error	Cannot import from file '*file name*' into table: Invalid table path '*table path*' format	Importing the data from file *file name* failed due to the structure or command table path/name *table path* being in the incorrect format
Error	Cannot import from file '*file name*' into table: No columns match those in the target table	The CSV-formatted import file *file name* has no columns defined that match those in the table to which the file is being imported; no data is added to the table from the file. Check the import file's column names
Error	Cannot import from file '*file name*' into table: Table type '*table type*' definition column name missing	Importing the data from file *file name* failed due to the table type definition *table type* is missing the column name. Add the missing column name
Error	Cannot import from file '*file name*' into table: Tag information missing	The CSV-formatted import file *path+file name* has no tag (e.g., _description_ or _column_names_) prior to the table information
Error	Cannot import from file '*file name*' into table: Too many/few table name and type inputs	The number of inputs following the _name_type_ tag in the the CSV-formatted import file *file name* is incorrect (should be two)
Error	Cannot import from file '*file name*' into table: Unknown table type '*table type*'	Importing the data from file *file name* failed due to the table type *table type* not existing in the project or being defined in the import file. Add the missing table type definition to the project or import file
Error	Cannot import from file '*file name*' into table: Unrecognized file type	Importing the data from file *file name* failed due to the file type not being recognized. The file extension must end in .csv, .xtce, or .eds
Error	Cannot import from file '*file name*' into table: XTCE conversion setup failed; cause '*error cause*'	Importing the data from file *file name* failed during creation of the XTCE XML marshaller/unmarshaller due to the specified cause
Error	Cannot import XTCE XML from file '*file name*'; cause '*error cause'*	Importing the project from file *file name* in XTCE XML format failed due to the specified cause
Error	Cannot load data from the custom values table	The attempt to load the table path(s), column name(s), and column value(s) in the custom values table matching the specified column name and column value failed. Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot load internal table '*table name*'	The attempt to load the data from internal table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot load table '*table name*'	The attempt to load the data from table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot load table members	The attempt to load the table and child table relations failed. Detail on the cause is logged in the event log
Error	Cannot locate backup file '*path+file name*'	The project database restore file *file name* cannot be found in the specified directory *path*
Error	Cannot locate event log file '*path+file name*'	The event log file *file name* cannot be found in the directory *path*
Error	Cannot modify data in table '*table name*'	The attempt to update the contents of table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot modify project data field(s)	The attempt to update the project description or project-level data field(s). Detail on the cause is logged in the event log
Error	Cannot obtain column order for table '*table name*'	The attempt to query the project database for the column order for table *table name*. Detail on the cause is logged in the event log
Error	Cannot obtain command information	The attempt to query the project database for the command information failed. This information is used to populate the Command Information dialog and command references input type selection item list. Detail on the cause is logged in the event log
Error	Cannot obtain comment for internal table '*table name*'	The attempt to query the project database for the comment on internal table *table name* failed. Detail on the cause is logged in the event log
Error	Cannot obtain comment for project database '*database name*'	The comment for the project database *database name* cannot be retrieved. Detail on the cause is logged in the event log
Error	Cannot obtain comment for table '*table name*'	The attempt to query the project database for the comment on table *table name* failed. Detail on the cause is logged in the event log
Error	Cannot obtain database version number	The database's version number cannot be obtained. Detail on the cause is logged in the event log
Error	Cannot obtain description for table '*table name*'	The attempt to query the project database __values table for the description of the table *table name* failed. Detail on the cause is logged in the event log
Error	Cannot obtain JDBC version number	The JDBC version number cannot be obtained. Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot open output file '*path+file name*'	The output file *file name* cannot be opened in the directory *path*. Check that the file permissions allow the user to read from this file and directory
Warning	Cannot parse import file '*path+file name*'	The JSON import file *path+file name* contains text that is not in the expected JSON format
Error	Cannot read backup file '*path+file name*'; cause '*error cause*'	The backup file *file name*, chosen to restore a project database, cannot be read for the reason *error cause*. Check that the file permissions allow the user so read from this file and directory
Warning	Cannot read event log file	The event log file cannot be read. Check that user has file read permissions for the file and directory
Error	Cannot read import file '*path+file name*'	The import .csv file *file name* cannot be read in the directory *path*. Check that the file permissions allow the user so read this file and directory
Error	Cannot read script file '*path+file name*'	The script file *file name* cannot be read in the directory *path*. Check that the file permissions allow the user so read this file and directory
Error	Cannot register database driver '*driver name*'	An error occurred registering the JDBC database driver *driver name*. This can be caused by setting an invalid server type
Error	Cannot release save point	A save point was created prior to executing one or more database commands. Following command execution, releasing the save point failed. Detail on the cause is logged in the event log
Error	Cannot rename project '*project name*'	An error occurred preventing renaming of the project *project name*. Detail on the cause is logged in the event log
Error	Cannot rename table '*table name*'	The attempt to rename table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot rename table type for table '*table name*'	The attempt to rename the table type for table *table name* in the project database failed. Detail on the cause is logged in the event log
Error	Cannot replace existing backup file '*path+file name*'	The project database backup file *file name* already exists in the directory *path*, but cannot be removed so as to be replaced by a new backup file of the same name. Check that the file permissions allow the user to write to this file and directory
Error	Cannot replace export file '*path+file name*'	The export .csv file *file name* already exists in the directory *path*, but cannot be removed so as to be replaced by a new file of the same name. Check that the file permissions allow the user to write to this file and directory

Type	Message	Cause
Error	Cannot replace output file '*path+file name*'	The output file *file name* already exists in the directory *path*, but cannot be removed so as to be replaced by a new file of the same name. Check that the file permissions allow the user to write to this file and directory
Error	Cannot replace script file '*path+file name*'	The script file *file name* already exists in the directory *path*, but cannot be removed so as to be replaced by a new file of the same name. Check that the file permissions allow the user to write to this file and directory
Error	Cannot respond to web server request	An error occurred in writing the output to the output stream for a request for data from the application via the web server. Detail on the cause is logged in the event log
Error	Cannot retrieve clipboard values; cause '*error cause'*	An error occurred in retrieving the values from the clipboard for a paste operation
Error	Cannot retrieve *list type* list	An error occurred retrieving the list of data *list type* from the project database. Detail on the cause is logged in the event log. The may be due to database corruption or a database server error
Error	Cannot revert changes to project	Following an update error on the current project another error prevented reverting any changes made to the database. Detail on the cause is logged in the event log
Error	Cannot set comment for project '*project name*'	The attempt to update the lock status, which is stored in the project database comment, for project '*project name*' failed. Detail on the cause is logged in the event log
Error	Cannot store internal table '*table name*'	The attempt to store the data to internal table *table name* in the project database failed. Detail on the cause is logged in the event log
Warning	Cannot store program preference values; cause '*error cause*'	The program preference keys could not be stored in the preference storage node due to the cited cause
Error	Cannot update comment for table '*table name*'	The attempt to update comment for table *table name* failed. Detail on the cause is logged in the event log
Error	Cannot update data fields	The attempt to update the data fields in the internal table (*__fields*) failed. Detail on the cause is logged in the event log
Error	Cannot update data types	The attempt to update the data types in a data table or the internal table (*__data_types*) failed. Detail on the cause is logged in the event log

Type	Message	Cause
Error	Cannot update input types	The attempt to update the input types in a data table or data field or the internal table (*__input_types*) failed. Detail on the cause is logged in the event log
Error	Cannot update macros	The attempt to update the macros in a data table or the internal table (*__macros*) failed. Detail on the cause is logged in the event log
Error	Cannot update project administrator(s)	The attempt to update the project administrator(s) in the project database comment failed. Detail on the cause is logged in the event log
Error	Cannot update table type '*type name*' <and table(s) '*table name(s)>*'	The attempt to update table type *type name* (and tables of that type, *table name(s)*, if any) in the project database failed. Detail on the cause is logged in the event log
Warning	Cannot write to event log	The event log file cannot be written. Check that user has file write permissions for the file and directory
Error	Cannot write to export file '*path+file name*'	An I/O error occurred while writing to the export file *file name* in the directory *path*
Error	Cannot write to script file '*path+file name*'	An I/O error occurred while writing to the script file *file name* in the directory *path*
Warning	Column '*column name*' expects a boolean value	The text pasted into column *column name* is non-boolean (true/false) and the column only displays boolean (in the form of a check box); the text is ignored
Warning	Column name '*column name*' already in use	The column name *column name* is already used in the table type being edited. A different column name must be chosen
Warning	Column name '*column name*' already in use (database)	The database converts the column names to one that is valid for use in PostgreSQL. The database form of the column names in the table type being edited must be unique. A different column name must be chosen
Warning	Column name '*column name*' already in use (hidden)	The column name *column name* is already used by a hidden column in the table type being edited. A different column name must be chosen
Warning	Column name '*column name*' matches a reserved word	The column name *column name* matches a PostgreSQL key word. A different column name must be chosen
Warning	Data field '*field name*' definition applicability type '*applicability type*' for owner '*owner name*' unrecognized in import file '*file name*'; continue?	The applicability type, *applicability type*, referenced in a data field definition with the name *field name* belonging to field owner *owner name* in the import file *file name* is not one of those recognized. Change the input type, or select **Ignore** or **Ignore all** to use the default input type ('All tables')

Type	Message	Cause
Warning	Data field '*field name*' definition input type '*input type*' for owner '*owner name*' unrecognized in import file '*file name*'; continue?	The input type, *input type*, referenced in a data field definition with the name *field name* belonging to field owner *owner name* in the import file *file name* is not one of those recognized. Change the input type, or select **Ignore** or **Ignore all** to use the default input type ('Text')
Warning	Data must be provided for column '*column name 1*' or column '*column name 2*' [row *row number*]	One or both columns *column name 1* and *column name 2* in the data type editor in row *row number* require a value, but both are empty. Enter a value in at least one of the columns
Warning	Data must be provided for column '*column name*' [row *row number*]	The column *column name* in the table type, data type, input type, macro, data field, reserved message ID, or access level editor in row *row number* requires a value, but is empty. Enter a value in the column
Warning	Data type name already in use	The data type name entered in the data type editor's User Name column is already in use by another data type. User-defined data type names must be unique. Alter the data type name to one not in use
Warning	Data type size must be a positive integer	The value entered for a data type's size is less than 1 or is not an integer. Enter a valid size value
Warning	Database connection parameter(s) missing	One or more or the server connection parameters, server type, server host, or user name, are missing. the **Change user** and **Server properties** commands are used to set these parameters
Error	Database query failed	A project database query executed from within a script using the getDatabaseQuery script data access method failed. The script association dialogs can also produce this error. Detail on the cause is logged in the event log
Warning	Directory name cannot be selected as the file name	A directory (folder) name is entered as the file name in the export dialog. Select a file and not a directory
Warning	Duplicate stream name	A data stream name netered in the rate parameter dialog matches one already in use. Data stream names must be unique
Warning	Enumeration '*enumeration'* format invalid in table '*table name*'; initial non-negative integer or separator character between enumeration value and label missing	One or more of the enumeration definitions in enumeration *enumeration* in table *table name* imported from an EDS or XTCE XML file does not have a non-negative integer as the first enumeration parameter or the character separating the enumeration value and label can't be identified. EDS and XTCE XML enumerations must be in the format specified in paragraph 4.5.6

Type	Message	Cause
Warning	Enumeration '*enumeration'* format invalid in table '*table name*'; separator character between enumerated pairs missing	The character separating each enumerated pair can't be identified in one or more of the enumeration definitions in enumeration *enumeration* in table *table name* imported from an EDS or XTCE XML file. EDS and XTCE XML enumerations must be in the format specified in paragraph 4.5.6
Warning	Environment variable override key '*key name*' has no corresponding value	The environment variable override key *key name* entered in the script manager or script executive dialog doesn't have a value associated with it. The key/value format is `<key1 = value1<,key2 = value2<,…>>>`
Error	Error obtaining metadata for internal table '*table name*'	An error occurred while attempting to read the metadata for the internal table *table name* during project database verification. The metadata provides information on the table's columns (number, names, and data types)
Error	Error obtaining metadata for table '*table name*'	An error occurred while attempting to read the metadata for the table *table name* during project database verification. The metadata provides information on the table's columns (number, names, and data types)
Error	Error obtaining project database '*database name*' metadata	An error occurred while attempting to read the metadata for project database *database name* during project database verification. The metadata provides information on the number of tables and their names
Error	Error verifying project database '*database name*' consistency	An error occurred while perform updates to project database *database name* internal tables during project database verification. Detail on the cause is logged in the event log
Command Line Error	Error: *argument* must be >= *minimum* and <= *maximum*	The command line argument *argument* expects a numeric value between the values *minimum* and *maximum*, inclusive
Command Line Error	Error: *argument* must be a color name or in the format '0x######' where '#' is a hexadecimal digit	The command line argument *argument* expects a recognized color name from the color map or a hexdecimal value preceded by '0x'
Command Line Error	Error: *argument* must be one of the following: *valid inputs*	The command line argument *argument* is provided an argument value that is not one of the valid inputs, *valid inputs*, for this command
Command Line Error	Error: *argument* not a number	The command line argument *argument* expects a numeric value which isn't provided
Command Line Error	Error: Import disabled; user lacks write access	The import command was executed by a user with read only access. This command is only available to users with read/write or admin access level.

Type	Message	Cause
Command Line Error	Error: mainsize width or height not a number, or too many/few values	The width or height contains a non-numeric (0-9) character, or other than 2 values are given
Command Line Error	Error: Missing argument for command '*command name*'	The command *command name* expects an argument immediately following the command but none is provided
Command Line Error	Error: Missing export file name and/or table path(s)	The export command is missing the export file name and/or the data table path(s)
Command Line Error	Error: Missing import file name	The import command is missing the import file
Command Line Error	Error: Unrecognized command '*command name*'	The command *command name* isn't a valid command for CCDD
Command Line Error	Error: Unrecognized delimiter for command '*command name*'	The command *command name* uses an unrecognized delimiter. Commands must begin with either '-' or '/'
Warning	Field name '*field name*' already in use	The data field *field name* is already in use for this table. Each field within a table must be unique. Alter the field name
Warning	Field size must be a positive integer	The value entered for a data field's size is less than 1 or is not an integer. Enter a valid size value
Warning	Field size must be less than or equal to *maximum width*	The value entered for a data field's size is greater than the maximum allowed, *maximum width* (determined by the program preference value **MaximumDataFieldLength**). The number of characters that can be entered into the field is not limited by this value, only the displayed width of the field. Enter a value less than the maximum
Error	File '*path+file name*' is not a backup file	The file chosen to restore a project database is not in the expected format. The file is either corrupted or the wrong file was chosen
Warning	Group name is already in use	The group name entered in the group name text field is already in use by another group or matches the pseudo-group's name, 'All tables'. Group names must be unique. Alter the group name to one not in use
Warning	Group name must be entered	The group name text field is empty. Enter a valid group name into the text field
Warning	ID interval must be a positive integer	The message ID interval value in the Assign Telemetry Messages or Assign Table Message IDs dialog is invalid. Enter a positive integer value
Warning	Illegal character(s) in association name	The association name in the script association manager dialog contains one or more illegal characters. Association names must begin with a letter or underscore and contain only letters, numerals, and underscores. Remove the illegal character(s)

Type	Message	Cause
Warning	Illegal character(s) in data type C type name	The C type name in the data type editor table cell contains one or more illegal characters. C type names can consist of multiple words, separated by one or more spaces, which must begin with a letter or underscore and contain only letters, numerals, and underscores (an ending asterisk is legal if the corresponding base type is 'pointer' or blank). Remove the illegal character(s)
Warning	Illegal character(s) in data type name	The user data type name in the data type editor table cell contains one or more illegal characters. Data type names must begin with a letter or underscore and contain only letters, numerals, and underscores. Remove the illegal character(s)
Warning	Illegal character(s) in macro name	The macro name in the macro editor table cell contains one or more illegal characters. Macro names must begin with a letter or underscore and contain only letters, numerals, and underscores. Remove the illegal character(s)
Warning	Illegal character(s) in project name	The project name text field contains one or more semi-colons. Remove the illegal character(s)
Warning	Illegal character(s) in table name '*table name*'	The table name text field contains one or more illegal characters. Table names must begin with a letter or underscore and contain only letters, numerals, and underscores. Remove the illegal character(s)
Warning	Incorrect number of columns indicated for table '*table name*' in the column order table for user '*user name*'	Detected during project database verification, the number of columns for table *table name* in the internal table __orders doesn't match the actual number of columns for that table's type. If updated the column order is reset to the default
Warning	Input type name already in use	The input type name entered in the input type editor's Type Name column is already in use by another input type (this includes the default input types as well). Input type names must be unique. Alter the input type name to one not in use
Warning	Input value out of range in table '*table name*' for column '*column name*'; must be greater than *minimum* and less than *maximum*	The minimum or maximum value entered in a cell of table *table name* in the column *column name* is outside the minimum and maximum values for an unsigned integer of the data type's size. Enter a value within the minimum and maximum boundaries

Type	Message	Cause
Warning	Internal table '*table name*' column '*column name*' data type mismatch (expected: '*expected type*', actual: '*actual type*')	Detected during project database verification, the data type for the column *column name* in the internal table *table name* is found to not be of the type expected for this column (e.g., an integer type is specified while the table shows a text type). If updated the data type is changed to the one expected
Warning	Internal table '*table name*' column *column index* name mismatch (expected: '*expected name*', actual: '*actual name*')	Detected during project database verification, the column indicated by its index is found to have a name other than the name expected for this column. If updated the name is changed to the one expected; however, the data in the column may be incorrect as well. For this case deleting the internal table (with loss of its data) may be necessary
Warning	Internal table '*table name*' has too many columns	Detected during project database verification, the internal table *table name* is found to have more columns than the number expected. If updated any extra columns are removed
Warning	Internal table '*table name*' is missing one or more columns	Detected during project database verification, the internal table *table name* is found to be missing one or more columns. If updated the table is deleted (with loss of its data)
Error	Invalid application parameter(s): using the default values instead	The application scheduler parameters stored in the project database internal table (*__app_scheduler*) comment are invalid. Default values replace these parameters. Detail on the cause is logged in the event log
Error	Invalid application scheduler applications detected; *number* removed	The application scheduler internal table (*__app_scheduler*) references applications that do not exist in the project database. The *number* invalid application references are removed
Warning	Invalid bit length in table '*table name*'; bit length exceeds the size of the data type	Either the bit length entered for a parameter in the data table *table name* is larger than the size, in bits, of the associated data type, or the data type changed to a smaller sized integer with a size in bits less than the current bit length. Decrease the bit length or choose a data type containing more bytes
Warning	Invalid characters in field '*field name*'; characters consistent with input type '*input type*' expected	The value in the data field *field name* text field contains characters that are inconsistent with the data field's input type, *input type*. Enter only characters matching the input type or change the input type

Type	Message	Cause
Warning	Invalid characters in message ID	The message contains an invalid character in the telemetry scheduler Scheduler ID column cell. Enter an ID in hexadecimal format (the leading '0x' is optional)
Warning	Invalid characters in message name	The message contains an invalid character in the telemetry scheduler Scheduler Message column cell. Enter a name beginning with an underscore or alphabetical character, and containing only alphanumeric and underscore characters
Warning	Invalid characters in separator field(s)	The variable path or data type/variable name separator characters entered in the Variable Path & Names dialog are not valid. The left and right bracket characters ([and]) are not allowed. Remove the invalid character(s) from the separator inputs
Warning	Invalid characters in table '*table name*' for column '*column name*'; characters consistent with input type '*input type*' expected	The value entered in a cell of table *table name* in the column *column name* does not match the expected input type, *input type*, as specified in the table type definition. Enter only characters matching the input type or change the input type
Warning	Invalid data type 'data type' is table '*table name*'; structure cannot reference itself or an ancestor	The data type pasted into the structure table *table name*'s data type cell matches the name of the structure or one of its ancestor structures. This causes a recursive reference, so isn't allowed. Change the data type to a valid structure reference or a primitive data type
Warning	Invalid export file name	The file name entered in the export dialog is missing or invalid. Enter a valid file name
Warning	Invalid input type in table '*table name*' for column '*column name*'; input type 'non-negative integer' expected	The minimum or maximum value entered in a cell of table *table name* in the column *column name* for an unsigned integer data type is less than zero. Enter a non-negative integer value
Warning	Invalid input type in table '*table name*' for column '*column name*'; input type 'integer' expected	The minimum or maximum value entered in a cell of table *table name* in the column *column name* for an integer data type is not an integer value. Enter an integer value
Warning	Invalid input type in table '*table name*' for column '*column name*'; input type 'float' expected	The minimum or maximum value entered in a cell of table *table name* in the column *column name* for a floating point data type is not a floating point value. Enter a floating point value
Warning	Invalid input value in table '*table name*' for column '*column name*'; the minimum must be less than or equal to the maximum	In table *table name* in the column *column name*, representing a minimum or maximum value, the minimum value is greater than the maximum value. Alter the minimum ro maximum such that the minimum is less than or equal to the maximum

Type	Message	Cause
Warning	Invalid input value in table '*table name*' for column '*column name*'; command argument names must be unique for a command	The command argument name entered in a cell of table *table name* in the column *column name* has a duplicate elsewhere in that column, and the cell values must be unique, as specified in the table type definition
Warning	Invalid input value in table '*table name*' for column '*column name*'; data type invalid or unknown in sizeof() call	The data type referenced in a sizeof() call entered in a cell of table *table name* in the column *column name* is either not a recognized primitive data type or structure name, or is a valid structure name but is the prototype of this structure table or one of its child structures, producing a recursive reference. Enter a valid data type name. Use Ctrl-S to display a pop-up dialog that allows selecting from the valid data types
Warning	Invalid input value in table '*table name*' for column '*column name*'; input value '*input value*' must be unique	The value, *input value*, entered in a cell of table *table name* in the column *column name* has a duplicate elsewhere in that column, and the cell values must be unique, as specified in the table type definition
Warning	Invalid message ID range; lower value must be <= upper value	The upper message ID range value is less than the lower value entered into the reserved message ID dialog. Correct the lower or upper value
Warning	Invalid message ID; hexadecimal range expected	The message ID range entered into the reserved message ID dialog doesn't match the expcted format: *start hexadecimal value <- end hexadecimal value>*. Enter a valid hexadecimal value or values
Error	Invalid rate parameter(s): using the default values instead	The rate parameters stored in the project database internal table (*__tlm_scheduler*) comment are invalid. Default values replace these parameters. Detail on the cause is logged in the event log
Warning	Invalid regular expression; cause '*cause'*	A project database or script search was attempted using an invalid regular expression (the check box allowing a regular expression must be selected in order for the search text to be evaluated as a regular expression). The reason is indicated by *cause*. Enter a valid regular expression and reattempt the search
Warning	Invalid regular expression; cause '*error cause*'	The regular expression entered in the input type editor's RegEx Match column is invalid for the cause indicated. Correct the regular expression

Type	Message	Cause
Warning	Invalid variable path in table '*table name*'; variable path already in use in another structure	In table *table name* the variable path entered in the column with input type 'Variable path' is a duplicate of that entered for another variable (in this or any other structure). The variable path must be unique. Alter the variable path to be unique or use the default path
Error	Invalid web server request	The request for data from the CCDD application via the web server is unrecognized; an unknown data stream name, incorrect number of parameters, or incorrect parameter type was passed to the telemetry scheduler request; or an error occurred while attempting to parse the data from the database for the request. Detail on the cause is logged in the event log
Warning	Link name is already in use	The link name entered in the link name text field is already in use by another link. Link names must be unique. Alter the link name to one not in use
Warning	Link name must be entered	The link name text field is empty. Enter a valid link name into the text field
Warning	Macro '*macro name*' contains a recursive reference	The macro value for the macro *macro name* in the macro editor contains a circular reference (a macro references itself). Alter the macro value to remove the circular reference
Warning	Macro name is already in use	The macro name entered in the macro editor's name column is already in use by another macro. Macro names must be unique. Alter the macro name to one not in use
Warning	Macro value is not consistent with macro usage in table(s) '*table name(s)*'	The macro value entered in the macro editor's value column does not match the input type of a column in one or more tables, *table name(s)*, where the macro is used. Alter the macro value to be consistent with the input type in every column for which the macro is referenced
Warning	Message ID is already in use	The message ID is a duplicate of another in the telemetry scheduler Scheduler ID column. Enter a unique message name
Warning	Message ID(s) already reserved	The message ID range entered into the reserved message ID dialog matches or encompasses an ID already reserved. Correct the range values so that no overlap occurs
Warning	Message name is already in use	The message name is a duplicate of another in the telemetry scheduler Scheduler Message column. Enter a unique message name
Warning	Message name must be entered	The message name is missing from the telemetry scheduler Scheduler Message column cell. Enter a valid message name

Type	Message	Cause
Warning	Message name pattern must be in the format: *startText*<0#>d<*endtext*> where *startText* and *endText* consist of alphanumeric characters and/or underscores, *startText* begins with a letter or underscore, and *#* is one or more digits. Note: *0#* and *endText* are optional	The message name pattern in the Assign Telemetry Messages dialog in not in the expected format. The pattern must contain only alphanumeric characters, contain a single '#' character, and begin with either an underscore or alphabetical character. Change the pattern to match the valid format
Warning	Message starting number must be an integer >= 0	The message starting number in the Assign Telemetry Messages dialog is invalid. Enter a positive integer value or zero
Warning	Missing or extra data type definition input(s) in import file '*file path+name*'; continue?	A data type definition is missing or has too many inputs in import file *file path+name*
Warning	Missing or extra input type definition input(s) in import file '*file path+name*'; continue?	An input type definition is missing or has too many inputs in import file *file path+name*
Warning	Missing or extra macro definition input(s) in import file '*file path+name*'; continue?	A macro definition is missing or has too many inputs in import file *file path+name*
Warning	Missing or extra reserved message ID definition input(s) in import file '*file path+name*'; continue?	A reserved message ID definition is missing or has too many inputs in import file *file path+name*
Warning	Missing table type name in import file '*file path+name*'; continue?	A table type definition is missing the table type name in import file *file path+name*
Warning	Must enter or select a script	No script is selected when the Add button is pressed in the script association manager dialog. Enter or select a script file
Warning	Must select a project to delete	No project is selected from the Delete Project dialog when the Delete button is pressed. Select one or more projects from the dialog or press the Cancel button
Warning	Must select a project to open	No project (other than the currently open one) is selected from the Open Project dialog when the Open button is pressed. Select a project from the dialog or press the Cancel button
Warning	Must select a project to unlock	No project is selected from the Unlock Project dialog when the Unlock button is pressed. Select a project from the dialog or press the Cancel button
Warning	Must select a script location	No folder is selected in which to save the script(s) retrieved from the project when the Retrieve button is pressed in the Retrieve Script dialog. Enter or select a script location, or press the Cancel button

Type	Message	Cause
Warning	Must select a script to delete	No script is selected from the Delete Script(s) dialog when the Delete button is pressed. Select a script from the dialog or press the Cancel button
Warning	Must select a script to retrieve	No script is selected from the Retrieve Script(s) dialog when the Retrieve button is pressed. Select a script from the dialog or press the Cancel button
Warning	Must select a script to store	No script is selected from the Store Script(s) dialog when the Store button is pressed. Select a script from the dialog or press the Cancel button
Warning	Must select a table from the tree	No table is selected from the table tree and is required for the action requested by the user. Select a table from the tree
Warning	Must select an export file name	No export file name is entered in the export dialog. Enter a valid file name
Warning	Must select an import file name	No import file name is entered in the import dialog. Enter a valid file name
Warning	Must select at least one data field	No data field is selected from the list of fields in the data field table editor selection dialog. Select at least one data field check box
Warning	Must select at least one structure table	No table is selected from the structure table tree in the padding dialog. Select at least one table
Warning	No data field exists	No data field is available to select in the data field table editor selection dialog
Warning	No other user exists	An attempt was made to change to another user when no other user exists in the server
Warning	No project exists for which user '*user name*' has access	The user *user name* does not have permission to access any of the project databases existing in the server. The user's permissions must be upgraded or a project database created for which the user has access
Warning	No role exists	No user or role exists in the server from which to choose
Warning	Password incorrect for user 'user name'	The password entered for user *user name* is invalid. Enter the correct password
Warning	Platform does not allow key press simulation	Copy, paste, and insert menu commands in the table and table type editors are handled by simulating the equivalent control key presses. The platform on which the application is running does not support this type of simulation. Use the actual key press sequences to perform the desired operation
Warning	Problem occurred when setting the look & feel to *look&feel*	An exception occurred while attempting to set the look & feel to the one selected. This can occur if the look & feel is not supported by the platform, or if there is a problem with access to the look & feel information

Type	Message	Cause
Error	Project '*database name*' restore failed	An error occurred preventing restoring project *project name*. Detail on the cause is logged in the event log
Warning	Project '*project name*' has no table type defined	The project database *project name* has no __types internal table or the __types table is empty. Create table types using the table type editor and store these in the project's database
Error	Project '*project name*' backup failed	An error occurred preventing backing up project *project name*. Detail on the cause is logged in the event log
Warning	Project '*project name*' has no scripts	The user attempted to retrieve a script from the project database *project name*, but the project does not have any scripts stored in it
Warning	Project '*project name*' has no tables	The project database *project name* contains no data tables. Create tables using the **Table** | **New** command
Warning	Project database name too long (*maximum length* characters maximum)	The project's database name, derived from the project name entered into the project name text field, exceeds the maximum allowed. The maximum length for a database name in PostgreSQL is 63 characters. Shorten the name to within the length limit
Warning	Project must be selected	No project is selected when renaming or copying a project database. Choose a project from the radio button list
Warning	Project name already in use	The project (database) name already exists on the server. Choose another name that does not match an existing project's database
Warning	Project name must be entered	The project name text field is empty. Enter a valid project name into the text field
Warning	Project owner must be selected	No owner is selected when creating a project database. Choose an owner from the radio button list
Warning	Project-level data field has missing or extra input(s) in import file '*file path+name*'; continue?	A project-level data field is missing or has too many inputs in import file *file path+name*. Add the missing inputs or delete the extraneous ones
Warning	Rate parameter values must be positive integers	The value in one or more rate parameter dialog input text fields contains a zero, negative, or non-integer value. Enter an integer value greater than or equal to 1 in each of the fields
Warning	Script file name missing	The script file name is missing in the export dialog. A file name must be entered if the check box indicating an external file is used is selected. Enter a valid script file name or deselect the check box

Type	Message	Cause
Warning	Search text cannot be blank	A project database or script search was attempted without a text string for which to search entered in the search dialog. Enter a text string prior to attempting a search
Warning	Server port must be a positive integer	The value entered into the server port field in the web server properties dialog is invalid. Enter a port number (positive integer value)
Warning	Server port must be blank or a positive integer	The value entered into the server port field in the PostgreSQL server properties dialog is invalid. Either clear the field or enter a port number (positive integer value)
Warning	Starting ID must be in the format <0x>#, where # is one or more hexadecimal digits	The starting message ID value in the Assign Telemetry Messages or Assign Table Message IDs dialog is invalid. Enter a hexadecimal value. The "0x" prefix is optional
Warning	System data field name, version, validation status, and/or classification missing	The Export XTCE dialog system data field name, version, validation status, and/or classification field is empty. Enter a valid value for each missing field
Warning	Table '*table name*' column '*column name*' data type is invalid (*data type*)	Detected during project database verification, column *column name* in table *table name* is found to have an invalid data type, *data type*. Updating replaces the data type with that from the table's type definition
Warning	Table '*table name*' column '*column name*' rows *row number 1* and *row number 2* have duplicate values	Detected during project database verification, the values in table *table name* on rows *row number 1* and *row number 2* in column *column name* are found to have the same value when the indicated column for this table's type is specified to contain only unique values. If updated the value in row *row number 2* is replaced with a blank
Warning	Table '*table name*' column name '*column name*' unrecognized in import file '*file path+name*'; continue?	The column *column name* for table *table name* in the import file *file path+name* doesn't not exist in the table type definition
Warning	Table '*table name*' contains a recursive reference to '*recursion table name*'	The table *table name* has the condition wherein the table *recursion table name* contains a reference to itself as a variable or as a variable in one of its child tables (or in one of their child tables, etc.). Remove the recursive table reference
Warning	Table '*table name*' has an unknown column ('*column name*')	Detected during project database verification, table *table name* is found to have a column *column name* that is not defined for this table's type. If updated the column is deleted

Type	Message	Cause
Warning	Table '*table name*' is an unknown type ('*table type*')	Detected during project database verification, the table type *table type* specified for table *table name* is not one of the defined table types. If updated the table is deleted
Warning	Table '*table name*' is missing column '*column name*'	Detected during project database verification, table *table name* is found to be missing a column *column name* that is defined for this table's type. If updated the column, with blank values for any rows, is added
Warning	Table '*table name*' printing failed; cause '*error cause*'	Output of the table *table name* to a printer or file was unsuccessful due to the cause specified. This can be due to the printer being offline
Warning	Table '*table name*' row *row number* column '*column name*' input type mismatch	Detected during project database verification, the value in row *row number*, column *column name* in table *table name* is found to have a value that is inconsistent with the input type specified in this table's table type for this column (e.g., text in an integer-only cell). If updated the value in the row and column indicated is replaced with a blank
Warning	Table '*table name*' row *row number* index mismatch	Detected during project database verification, row *row number* in table *table name* is found to have the wrong row index. Row indices, stored in a hidden column, start at 1 for the first row and increment sequentially for each additional row. If updated the row indices are set to the expected values
Warning	Table '*table name*' variable '*variable name*' array member *array index* array size doesn't match the array definition	Detected during project database verification, in table *table name* the array member *variable name*[*array index*] is found to have a value in the array size column that differs from that in the array's array definition. If updated the array size for the specified array member is changed to match the array definition
Warning	Table '*table name*' variable '*variable name*' array member *array index* data type doesn't match the array definition	Detected during project database verification, in table *table name* the array member variable name is found to have a value in the data type column that differs from that in the array's array definition. If updated the data type for the specified array member is changed to match the array definition
Warning	Table '*table name*' variable '*variable name*' has an extra array member	Detected during project database verification, in table *table name* the array variable *variable name* is found to have more members than its array size allows. If updated any extra array member rows are deleted

Type	Message	Cause
Warning	Table '*table name*' variable '*variable name*' is missing array member *array index*	Detected during project database verification, in table *table name* the array variable *variable name* is found to have fewer members than its array size allows. If updated any missing array member rows are added
Warning	Table '*table name*' variable '*variable name*' is missing the array definition	Detected during project database verification, in table *table name* the array member variable name is found to have no accompanying array definition. If updated the missing array definition row is added
Warning	Table *'table name*' has missing or extra data field input(s) in import file '*file path+name*'; continue?	A data field definition in table *table name* is missing or has too many inputs in import file *file path+name*
Error	Table export completed with errors	An error occurred when attempting to export one or more tables to a file. A separate error dialog appears describing the specific error; this error appears at the end of the export operation
Warning	Table name 'table name' cannot begin with '__'	The table name, *table name*, entered into the table name text field begins with a double underscore. The double underscore prefix is reserved for use by the application to designate internal tables in the project database. Alter the table name to meet the table naming constraints
Warning	Table name '*table name*' is a duplicate	The table name, *table name*, appears more than once in the list of new table names entered in the table name text field. Table names must be unique. Alter the table name to one not in use
Warning	Table name '*table name*' is already in use	The table name, *table name*, entered in the table name text field is already in use by another table. Table names must be unique. Alter the table name to one not in use
Warning	Table name '*table name*' matches a primitive data type	The table name, *table name*, entered in the table name text field matches a primitive data type's name (e.g., uint32, float). Alter the table name to meet the table naming constraints
Warning	Table name 'table name' matches a reserved word	The table name, *table name*, entered into the table name text field matches that of a reserved word in PostgreSQL. Alter the table name to meet the table naming constraints
Warning	Table name '*table name*' too long (*maximum* characters maximum)	The table name, *table name*, entered into the table name text field exceeds the maximum allowed (*maximum*). The maximum length for a table name in PostgreSQL is 63 characters. Shorten the name to within the length limit
Warning	Table name must be entered	The table name text field is empty. Enter a valid table name into the text field

Type	Message	Cause
Warning	Table type '*table type*' definition has missing or extra input(s) in import file '*file path+name*'; continue?	The table type definition *table type* is missing or has too many inputs in import file *file path+name*. Add the missing inputs or delete the extraneous ones
Warning	Table type '*table type*' definition input type '*input type name*' unrecognized in import file '*file path+name*'; continue?	The table type definition *table type* references an input type *input type name* that isn't a recognized input type in import file *file path+name*. Change the input type name to one of those recognized by the application. If the error is ignored then the invalid input type defaults to Text
Warning	Table type '*table type*' has missing or extra data field input(s) in import file '*file path+name*'; continue?	The table type definition *table type* is missing or has too many data field inputs in import file *file path+name*. Add the missing inputs or delete the extraneous ones
Warning	Table type must be selected	No table type is selected from the list. Select a table type
Warning	Table type name is already in use	The table type entered in the table type name text field is already in use by another table type. Table type names must be unique. Alter the table type name to one not in use
Warning	Table type name must be entered	The table type name text field is empty. Enter a valid table type name into the text field
Warning	Table(s) not exported '*table name(s)*'; output file already exists or file I/O error	The table(s) *table name(s)* selected for exportation were skipped due to the output file already existing and the option to overwrite existing files was not selected, or that a file I/O error occurred (for example, insufficient file permission in the target folder)
Warning	Table(s) not imported '*table name(s)*'; table already exists	The table(s) *table name(s)* selected for importation were skipped due to the table already existing and the option to overwrite existing tables was not selected
Warning	Tables of type 'Structure *or* Command' may not have more than one column with input type(s): *input type*[, *input type 2*[, …]]	In the table editor for a table type representing a 'Structure' or 'Command', one or more column definition input types (*input type*[, *input type 2*[, …]]) are used multiple times, but must be unique for this table type (e.g., input type 'Variable name' in a structure table). This can occur if these input types are assigned before the table has all the columns necessary to define the table type as a structure or command, then the final input type necessary to make the type a structure/command is assigned. Assign different input types to the affected row(s), or delete these row(s), then assign the input type

Type	Message	Cause
Warning	The value for '*preference name*' cannot be blank	The value for the size or spacing preference '*preference name*' in the Preferences dialog is empty. Enter a value between the minimum and maximum (inclusive)
Warning	The value for '*preference name*' is outside allowable limits	The value for the size or spacing preference '*preference name*' in the Preferences dialog is outside the minimum and maximum range. Enter a value between the minimum and maximum (inclusive)
Warning	The value for '*preference name*' must be a positive integer	A negative value was entered for the size or spacing preference '*preference name*' in the Preferences dialog. Enter a value between the minimum and maximum (inclusive)
Warning	Unknown input type '*text'*	The text *text* pasted into the Table Type Editor's Input Type column does not match a known input type; the text is ignored
Warning	Unknown internal table '*table name*'	Detected during project database verification, the table *table name* is found to have a name that indicates it is an internal table, but it is not one of the recognized internal tables. If updated the table is deleted
Warning	Unrecognized association name '*association name*'	The script association name *association name* entered for the command line **execute** command is not a valid association for the current project. Check the association name spelling
Warning	User name already in use	The user name selected in the access level manager **User Name** column matches that in another row. Each user may be assigned only one access level. Select another user name from the drop down menu or change the access level for the row that already contains the user name
Warning	User name must be entered	The user name field in the database login dialog is empty. Enter a valid user name into the text field. The user name field is only present if a connection to the database server cannot be established; otherwise a list of radio buttons representing the user list is displayed
Warning	User's guide '*file name*' cannot be opened; cause 'Desktop class unsupported'	The CCDD user's guide file cannot be opened. This is due to the Java Desktop class not being available in the operating system
Warning	User's guide '*file name*' cannot be opened; cause 'file I/O error or no application registered to open .pdf files'	The CCDD user's guide file cannot be opened. This is due to either a file I/O error or having no application registered in the operating system to open .pdf files (the help file is in PDF format)

Type	Message	Cause
Warning	User's guide '*file name*' cannot be opened; cause 'file missing'	The CCDD user's guide file cannot be opened. This is due to the file not being located in the CCDD start-up folder
Error	Web server failed to start	The attempt to start the embedded Jetty web server failed. Detail on the cause is logged in the event log
Error	Web server failed to stop	The attempt to stop the embedded Jetty web server failed. Detail on the cause is logged in the event log

Appendix E. Program Notes

Appendix E.1. Key reference

The special keys and their contexts/actions are described below.

←→↑↓ The left, right, up, and down arrow keys move table cell selection from the currently highlighted cell to the cell to the left, right, above, or below respectively. If the bottom (top) of the table is reached then the down (up) arrow traverses to the next component within the GUI entity as with the Tab (Shift-Tab) key. If in edit mode the left (right) key repositions the text cursor one character to the left (right) of the current cursor position until the left (right) end of the text string is reached; the up and down arrow keys have no effect while in edit mode.

Alt-Enter If editing a table cell with an input type that supports multiple lines, a line break is inserted into the table cell at the current text insertion point, replacing any selected character(s).

Ctrl-A Selects all cells in the table that currently has the focus.

Ctrl-Delete Deletes the currently selected row(s).

Ctrl-C Copies the contents of the selected table cell(s) to the clipboard buffer.

Ctrl-E Expands (if collapsed) or collapses (if expanded) the currently selected table or variable tree node(s). If multiple nodes are selected then the state of the uppermost one determines which action is taken for all of the selected nodes.

Ctrl-F Opens the **Search Session Event Log** dialog if pressed while the main application window has the focus (same as selecting the main window **File** | **Search log** command). Opens the table **Search** dialog if pressed while a table or table type editor dialog is open and has the focus (same as selecting editor's **File** | **Search** command).

Ctrl-I Pastes the contents of the clipboard buffer to the table cell(s). New rows are inserted to contain the pasted data.

Ctrl-M When pressed while in edit mode in a data table cell a pop-up list appears showing the macro or macros that are allowed in the cell (no pop-up appears if none of the defined macros is appropriate, or if no macros are defined). The chosen macro is inserted into the table cell at the current text insertion point, replacing any selected character(s) and bounded by the macro identifier characters.

Ctrl-Shift-M Replaces every macro reference and sizeof() call in the current table or macro editor value column with its corresponding text string, evaluated as a mathematical expression if applicable. Releasing the keys restores the macro names.

Ctrl-S When pressed while in the edit mode in a data table cell a pop-up list appears displaying the primitive data types and prototype structure names. If the data table represents a structure then only those prototype structures that can be used as a variable data type are displayed. When pressed while in the macro editor table a pop-up list appears displaying all primitive data types and prototype structure names. When pressed while in the data type editor's **Type Name** or **C Name** cells, and the corresponding **Base Type** is either blank or 'pointer', a pop-up list appears displaying all of the prototype structure names. The chosen data type name is inserted into the table cell.

Ctrl-V Pastes the contents of the clipboard buffer to the table cell(s), overwriting the current contents of the cell(s).

Delete When not editing a cell deletes the contents of the currently selected table cell(s). If in edit mode the Delete key removes the character immediately to the left of the text cursor.

End Changes the table cell selection to the leftmost column. If in edit mode the End key repositions the text cursor to the end of the text in the input cell or field.

Enter Enters edit mode when pressed while an editable table cell is selected. If in edit mode then the cell text is entered into the cell (following any validation) and the next editable cell to the right is placed in edit mode (if the rightmost column is reached then the leftmost cell in the row below is used; after the last row is reached the first row is used). If a cell containing a check box gets the focus then pressing Enter toggles the check box state rather than traversing to the next editable cell.

Escape Exits edit mode in a table cell or data field, restoring the original contents of the cell or field. Removed a pop-up list (macro) if displayed without making a selection.

Home Changes the table cell selection to the rightmost column. If in edit mode the Home key repositions the text cursor to the beginning of the text in the input cell or field.

Insert Inserts a new, empty row into the table below the row with the currently selected cell(s). If in edit mode the Insert key adds a space character at the text cursor location and moves the cursor immediately after the inserted space.

Page Down Scrolls the table one page down from its current position. Changes the cell selection to the currently selected column, with the row one page down from its previous position.

Page Up Scrolls the table one page up from its current position. Changes the cell selection to the currently selected column, with the row one page up from its previous position.

Shift-Delete Replaces the selected cell(s) value with that from the corresponding cell value in the tables' prototype. See paragraph 4.9.3.2.2.8.2 for more details.

Space Enters edit mode when pressed while an editable table cell is selected, then inserts a space in the cell (following any existing text).

Tab Changes the focus within the current GUI entity to its next component. Pressing Shift-Tab traverses the components in the opposite direction.

Appendix E.2. Program preferences

The program preferences are stored in a location dependent on the operating system and are updated as needed by the CCDD application. For example, the Windows operating system stores the preferences in the system registry under the key name:

```
HKEY_CURRENT_USER\Software\JavaSoft\Prefs\/C/C/D/D
```

In Linux the preferences are stored in the file:

```
/<user home directory>/.java/.userPrefs/CCDD/prefs.xml
```

Many of these preference values may be changed via the **Preferences** dialog; see paragraph 4.9.1.7. The preference keys and associated descriptions are provided below.

General

Database	The name of the project database that was connected to most recently.
DatabaseBackupPath	The full path to the folder to (from) which a project database was most recently backed up (restored). Used to set the initial path in the project database backup and restore dialogs.
HideDataType	Determines if the data type is excluded or included when displaying the variable path in the **Show variables** dialog and in structure tables containing a column with the input type 'Variable path'. The default is **false**.
LogFilePath	The full path name for the location where an event log was most recently opened for reading. This is not necessarily the path of the current session log. Used to set the initial path in the read log dialog.
PaddingAlignment	The current padding variable byte alignment value. The default is **4**. The value must be a power of 2.
PostgreSQLServerHost	The name of the PostgreSQL server host that was connected to most recently.
PostgreSQLServerPort	The PostgreSQL server port number of the server that was connected to most recently.
ScriptPath	The full path to the folder to (from) which a script was most recently retrieved from (stored in) the project database. Used to set the initial path in the Script storage and retrieval dialogs.
SearchStrings	Text string containing the previous search dialog searches, stored as a single string separated by special delimiter characters, and used for auto-completing input in the search dialogs' search text field. Once the maximum number of retained search strings is reached further searches cause the oldest search string to be removed so that the latest one can be added.
ServerStrings	Text string containing the previously entered server names, stored as a single string separated by special delimiter characters, and used for auto-completing input in the **Database Server** dialog's **Host** field. Once the maximum number of retained server names is reached further entries cause the oldest name to be removed so that the latest one can be added.
TableExportPath	The full path to the folder to (from) which a data table was most recently exported (imported). Used to set the initial path in the data table import and export dialogs.
TypeNameSeparator	The character(s) in a variable path used to separate the data type and variable name in the **Show variables** dialog and in structure tables containing a column with the input type 'Variable path'. The default is an underscore (_).
User	The name of the latest user to attempt a server connection.

VariablePathSeparator	The character(s) in a variable path used to separate the root structure table from a child variable, and a parent structure variable from a child variable in the **Show variables** dialog and in structure tables containing a column with the input type 'Variable path'. The default is an underscore (_).
WebServerPort	The web server port number of the server that was connected to most recently.

L&F

LookAndFeel	The name of the selected "look and feel" that governs the application's overall appearance.

Font

DataTableCellFont	Data table and table type editor cell font.
DialogButtonFont	Dialog box button label font.
InputFieldTextFont	Font used when inputting text into dialog input fields, description fields, and data fields.
LabelBoldFont	Font generally used for labels in dialogs.
LabelPlainFont	Font used for labels in dialogs for non-emphasized text. Also used in the telemetry and applications scheduler **Options** lists and **Assigned Applications** list.
MenuItemFont	Menu and sub-menu item font.
OtherTableCellFont	Table cell font for non-data tables (e.g., event log).
TableHeaderFont	Table column name font.
ToolTipFont	Font for use when displaying tool tips. Some look & feels ignore changes to the tool tip font.
TreeNodeFont	Font used when displaying tree node labels, such as in a table tree, link tree, etc.

Color

DataTypeTextColor	Text color for the data type portion structure and primitive variable name in a table or variable tree.
FocusBackgroundColor	Background color for a table cell that has the input focus.
InputBackgroundColor	Color used for the background in dialog input fields, description fields, and data fields.
InputDisabledBackgroundColor	Color used for the background in dialog input fields, description fields, and data fields that are currently non-editable.
InputTextColor	Color used for the text in dialog input fields, description fields, and data fields.

InvalidTextColor	Text color for invalid table rows. Used in the telemetry and application scheduler tables to denote messages and time slots which are valid for the selection option.
LabelTextColor	General color for text labels.
PaddingBackgroundColor	Background color for a padding variable cell.
ProtectedBackgroundColor	Background color for a protected (non-editable) table cell.
ProtectedTextColor	Text color for a protected (non-editable) table cell.
RequiredBackgroundColor	Background color for table cells and input fields that are required.
SelectedBackgroundColor	Background color for a table's cells when the cell is selected.
SelectedTextColor	Text color for a table's cells when the cell is selected.
SpecialLabelTextColor	Text color for a group of components in a dialog.
TableAlternateBackgroundColor	Background cell color for even numbered rows when the cell is not selected.
TableBackgroundColor	Table cell background color for odd numbered rows when the cell is not selected.
TableGridColor	Color for a table's grid lines. Log table grid lines are not colored (i.e., the log table uses the table background color between individual cells).
TableTextColor	Table cell text color when the cell is not selected.
TextHighlightColor	Color used to highlight macros and matching search text.
ToolTipBackgroundColor	Background color for tool tip pop-ups. Some look & feels ignore changes to the tool tip background color.
ToolTipTextColor	Text color for tool tip pop-ups. Some look & feels ignore changes to the tool tip text color.
TypeRequiredBackgroundColor	Background color for a cell in a table type column definition that is required to define the type.
ValidTextColor	Text color for valid table rows. Used in the telemetry and application scheduler tables to denote messages and time slots which are valid for the selection option.

Size

InitalViewableComponentRows	Number of rows of radio buttons or check boxes to display initially when using the CcddDialogHandler class methods for creating a radio button or check box selection panel.
InitialViewableDataTableRows	Maximum number of rows to display when a data table or table type editor is opened. The table may be resized afterwards to display fewer or more rows.

InitialViewableTableRows Maximum number of rows to display when a non-data table is opened. The table may be resized afterwards to display fewer or more rows.

MaximumConversionLists Maximum number of variable name conversion lists to maintain in memory. A conversion list is created when a full variable name, with path, is requested using separators other than those used in a previous request. When the maximum number of lists is reached an earlier list (the second one created) is removed to make room for the new one.

MaximumDataFieldLength Maximum character length for displaying a data field. Note that this is for display purposes only; the number of characters entered into the field may exceed this value.

MaximumDialogLineLength Maximum number of characters to display per line in a dialog message. The dialog text is wrapped for line lengths greater than this value.

MaximumDialogMessageLength Maximum number of characters to display in a dialog message. This value only is applied to dialogs displaying a list of tables in the event the number of tables is large.

MaxImportedTabRows Maximum number of tab rows a single editor dialog when importing tables. This value is used to prevent the number of tab rows from growing so large that the table content is no longer visible.

MaximumInitialTableCellWidth Maximum pixel width for a table column when the table is initially displayed. The column size may be changed afterwards.

MaximumLogMessageLength Maximum number of remembered search strings.

MaximumServerTimeout Number of seconds allowed to validate the PostgreSQL server connection.

MaximumToolTipLineLength Maximum number of characters to display per line in a tool tip.

MaximumViewableListRows Maximum number of items to display at one time in a combo box list. A scroll bar appears if more items than this are in the list.

MinimumDialogWidth Minimum dialog window width, in pixels. Any dialog displayed is this width, or larger if the contents dictate.

MinimumWindowHeight Minimum frame window height, in pixels. Examples of frame windows include the data table editors and table type editor. The frame window width may not be resized below this value.

MinimumWindowWidth Minimum frame window width, in pixels. Examples of frame windows include the data table editors and table type editor. The frame window height may not be resized below this value.

NumberOfRememberedSearches Maximum number of search strings to store. These are used for auto-completing input in the search dialogs' search text field. Once the maximum number of search strings is reached further searches cause the oldest search string to be removed so that the latest one can be added.

NumberOfRememberedServers Maximum number of server names to store. These are used for auto-completing input in the **Database Server** dialog's **Host** field. Once the maximum number of server names is reached further entries cause the oldest name to be removed so that the latest one can be added.

Spacing

ButtonGap Number of pixels between each button in a dialog box.

ButtonPad Minimum number of pixels around the perimeter of a dialog box's button grouping.

CellHorizontalPadding Number of pixels added to either side of the text in a table cell.

CellVerticalPadding Number of pixels added to above and below the text in a table cell.

DialogBorderPadding Number of pixels between the dialog box contents and the dialog's frame.

DialogIconPadding Number of pixels between the icon and text message in a message dialog box.

HeaderHorizontalPadding Number of pixels added to the width of a table's column header text. The padding is split equally between each side of the header text. This padding provides room for the column sort arrow.

HeaderVerticalPadding Number of pixels added to the height of a table's column header text. The padding is split equally between the top and bottom of the header text.

InputFieldPadding Number of pixels added to the each side of the text in an input field.

LabelHorizontalSpacing Defines the horizontal spacing between a text label and an adjacent component, in pixels.

LabelVerticalSpacing Defines the vertical spacing between a text label and an adjacent component, in pixels.

Other

EDSSchemaLocationURL The URL for the EDS schema location. This value is used to set the `JAXB_SCHEMA_LOCATION` property in the marshalled XML output when exporting project data in EDS format.

EnvironmentVariableOverride Environment variable override key/value pairs in the format <key=value<,…>>. The override values are used to expand variables in the script paths when executing script associations via the script manager or script execution dialogs; these overrides are not used when executing via the command line `execute` command.

XTCESchemaLocationURL The URL for the XTCE schema location. This value is used to set the `JAXB_SCHEMA_LOCATION` property in the marshalled XML output when exporting project data in XTCE format.

Appendix E.3. CCDD class files

Following is a list and description of the CCDD application's Java class files.

CcddApplicationParameterDialog.java	Dialog for assigning the application scheduling parameters. The dialog is built on the CcddDialogHandler class.
CcddApplicationParameterHandler.java	Class that handles retrieval from and storage to the project database of the application scheduling parameter values.
CcddApplicationSchedulerDialog.java	Dialog for assignment of applications to time slots. The dialog is built on the CcddDialogHandler class and implements the CcddSchedulerDialogInterface class.
CcddApplicationSchedulerInput.java	Class for handling application selection in the application scheduler dialog. This class implements the CcddSchedulerInputInterface class.
CcddApplicationSchedulerTableHandler.java	Class for handling CFS application scheduler table output.
CcddAssignmentTreeHandler.java	Class that handles the variable assignment tree in the telemetry scheduler dialog. This class is an extension of the CcddInformationTreeHandler class.
CcddAssignMessageIDDialog.java	Dialog for automatic assignment of message IDs to data tables or telemetry messages. The dialog is built on the CcddDialogHandler class.
CcddBackgroundCommand.java	Class for generically handling execution of code in a background thread.
CcddButtonPanelHandler.java	Generic utility class for creating and handling button panels in the dialogs and frames created within the application.
CcddClassesComponent.java	Collection of common classes used by other CCDD classes. These classes, in general, override existing Java component classes or introduce new ones.
CcddClassesDataTable.java	Collection of common classes used by other CCDD classes. These classes, in general, are used to manipulate and contain information with respect to the data tables.
CcddCommandDialog.java	Dialog for the user to view the project's commands (including the command's name, code, table, and argument names). The dialog is built on the CcddDialogHandler class.
CcddCommandHandler.java	Class for building a list of project commands (including the command's name, code, table, argument names, and number of arguments). This is used for populating the the **Command reference** input type selection item list and by the script data access methods.

CcddCommandLineHandler.java	Class for reading and executing the command line options.
CcddCommonTreeHandler.java	Class containing tree handling methods common to all other trees used in the application. This class is an extension of the JTree class.
CcddConstants.java	Class containing constant values used by the other classes.
CcddCopyTableHandler.java	Class for handling copy table operations.
CcddCSVHandler.java	Class for handling import and export of data tables in CSV format. This class implements the CcddImportExportInterface class.
CcddDataTypeEditorDialog.java	Class for handling data type editing. The dialog is built on the CcddDialogHandler class.
CcddDataTypeHandler.java	Class for handling data type operations.
CcddDbCommandHandler.java	Class for handling database commands.
CcddDbControlHandler.java	Class containing the methods for connecting to, creating, copying, renaming, and deleting project databases.
CcddDbManagerDialog.java	Dialog for the user to set the connection parameters to the database, and for creating, copying, renaming, and deleting databases. The dialog is built on the CcddDialogHandler class.
CcddDbTableCommandHandler.java	Class containing the methods for creating, altering, copying, renaming, and deleting the database tables.
CcddDbVerificationHandler.java	Class that executes the database information consistency check.
CcddDialogHandler.java	Generic utility class for creating and handling all of the dialogs created within the application.
CcddDuplicateMsgIDDialog.java	Dialog displaying a table containing duplicate message IDs and their owners. The dialog is built on the CcddDialogHandler class.
CcddEDSHandler.java	Class for handling import and export of data tables in EDS XML format. This class implements the CcddImportExportInterface class.
CcddEventLogDialog.java	Class for displaying and updating the session and stored event logs. The dialog is built on the CcddFrameHandler class.
CcddFieldEditorDialog.java	Class for handling data field operations. The dialog is built on the CcddDialogHandler class.
CcddFieldHandler.java	Class for handling the data field editor.

CcddFieldTableEditorDialog.java	Dialog for inspecting and assigning values to data input fields. The dialog is built on the CcddDialogHandler class.
CcddFileIOHandler.java	Class containing file input and output methods (project database backup and restore, table import and export, script storage and retrieval).
CcddFrameHandler.java	Generic utility class for creating and handling all of the frame windows created within the application.
CcddGroupHandler.java	Class for handling table grouping operations.
CcddGroupManagerDialog.java	Dialog for the user to create, alter, or delete table groups. The dialog is built on the CcddDialogHandler class.
CcddGroupTreeHandler.java	Class containing the methods for creating and manipulating a table group tree. This class is an extension of the CcddInformationTreeHandler class.
CcddImportExportInterface.java	Class that defines the interface for data table import and export classes.
CcddImportSupportHandler.java	Class containing support methods for classes based on the CcddImportExportInterface class. The support methods handle validation and addition of table types and data fields, and for obtaining the user's response to a non-fatal error condition. Classes utilizing these support methods must extend this class.
CcddInformationTreeHandler.java	Generic utility class for manipulating information trees. This class is an extension of the CcddCommonTreeHandler class.
CcddInputFieldPanelHandler.java	Class for creating the table editor panel in which a table, description, and data fields are displayed.
CcddJSONHandler.java	Class for handling import and export of data tables in JSON format. This class implements the CcddImportExportInterface class.
CcddJTableHandler.java	Generic utility class for creating and handling all of the tables created within the application, including the data, type, and field tables. This class is an extension of the JTable class.
CcddKeyboardHandler.java	Class for controlling keyboard input and implementing special key sequence actions.
CcddLinkHandler.java	Class containing methods to manipulate variable linkages.
CcddLinkManagerDialog.java	Dialog for the user to create, modify, or delete variable links, and to assign variables to the links. The dialog is built on the CcddDialogHandler class.

CcddLinkManagerHandler.java	Class for handling interactions with the variable links for a specific data stream.
CcddLinkTreeHandler.java	Class containing the methods for creating and manipulating a variable link tree. This class is an extension of the CcddInformationTreeHandler class.
CcddMacroEditorDialog.java	Dialog for the user to create, modify, or delete macros and macro values. The dialog is built on the CcddDialogHandler class.
CcddMacroHandler.java	Class for handling macro operations.
CcddMain.java	The CCDD main application class handles flow and execution of the menu bar items.
CcddMathExpressionHandler.java	Class for evaluating simple mathematical expressions.
CcddMessageIDDialog.java	Dialog displaying all message ID names and associated message ID values. The dialog is built on the CcddDialogHandler class.
CcddMessageIDHandler.java	Class used to determined which message IDs are currently used in tables, data fields, and telemetry messages, and to determine if any references are duplicated.
CcddPaddingAlignmentDialog.java	Dialog displaying a slider control that allows selection of the variable padding byte alignment value. The dialog is built on the CcddDialogHandler class.
CcddPaddingVariableHandler.java	Class that adds/updates or removes the padding variables.
CcddPatchHandler.java	Class used to contain code to update the project database when a schema change is made. The code is written to execute only if the database has not already been updated.
CcddPreferencesDialog.java	Class that creates and manages the Preferences dialog used for altering the application's look & feel, fonts, colors, size values, and spacing values. The dialog is built on the CcddDialogHandler class.
CcddProjectFieldDialog.java	Class that creates and manages project-level data fields. The dialog is built on the CcddDialogHandler class.
CcddRateParameterDialog.java	Dialog for assigning the telemetry sample rate parameters. The dialog is built on the CcddDialogHandler class.
CcddRateParameterHandler.java	Class that handles retrieval from and storage to the project database of the rate parameter values, and calculation of the sample rates based on the rate parameters.

CcddReservedMsgIDEditorDialog.java	Dialog for the user to create, modify, or delete reserved message ID and ID ranges and descriptions. The dialog is built on the CcddDialogHandler class.
CcddReservedMsgIDHandler.java	Class for handling reserved message ID operations.
CcddSchedulerDbIOHandler.java	Class for handling project database input and output operations for the applications and telemetry schedulers.
CcddSchedulerDialogInterface.java	Class that defines the interface for the application and telemetry scheduler dialog classes.
CcddSchedulerEditorHandler.java	Class that handles the Scheduler table within the application (for time slots) and telemetry (for messages) scheduler dialogs.
CcddSchedulerHandler.java	Class that manages the application and telemetry scheduler dialogs, including transfer of information between the trees and lists.
CcddSchedulerInputInterface.java	Class that defines the interface for application and telemetry scheduler input.
CcddScriptDataAccessHandler.java	Class containing the methods whereby scripts can access the project database information.
CcddScriptDataAccessHandlerStatic.java	Class containing the static method references to the methods in the CcddScriptDataAccessHandler class.
CcddScriptExecutiveDialog.java	Dialog for the user to select script associations to execute. The dialog is built on the CcddDialogHandler class.
CcddScriptHandler.java	Class that handles obtaining the table data and executing the associated script.
CcddScriptManagerDialog.java	Dialog for the user to associate scripts and data tables. The dialog is built on the CcddDialogHandler class.
CcddScriptStorageDialog.java	Dialog for the user to select script files to store to or retrieve from the database. The dialog is built on the CcddDialogHandler class.
CcddSearchDialog.java	Dialog for the user to perform text string searches of the project database data tables and stored scripts. The dialog is built on the CcddDialogHandler class.
CcddSearchHandler.java	Class that handles event log, table, and script searches.
CcddServerPropertyDialog.java	Dialog for changing the user name and password, and the PostgreSQL server host and port. The dialog is built on the CcddDialogHandler class.
CcddTableEditorDialog.java	Class for handling data table editing; displays instances of CcddTableEditorHandler. The dialog is built on the CcddEditorPanelHandler class.

CcddTableEditorHandler.java	Class that handles editing of a specific data table. This class is an extension of the CcddInputFieldPanelHandler class.
CcddTableManagerDialog.java	Dialog for the user create, edit, copy, rename, and delete data tables. The dialog is built on the CcddDialogHandler class.
CcddTableTreeHandler.java	Class containing the methods for creating and manipulating a data table tree. This class is an extension of the CcddCommonTreeHandler class.
CcddTableTypeEditorDialog.java	Class for handling table type editing; displays instances of CcddTableTypeEditorHandler. The dialog is built on the CcddEditorPanelHandler class.
CcddTableTypeEditorHandler.java	Class that handles the commands associated with a specific table type editor. This class is an extension of the CcddInputFieldPanelHandler class.
CcddTableTypeHandler.java	Class for handling interactions with table types.
CcddTableTypeManagerDialog.java	Dialog for the user to create, edit, copy, rename, and delete table types. The dialog is built on the CcddDialogHandler class.
CcddTelemetrySchedulerDialog.java	Dialog for assignment of variables to telemetry messages. The dialog is built on the CcddDialogHandler class and implements the CcddSchedulerDialogInterface class.
CcddTelemetrySchedulerInput.java	Class for handling variable selection in the telemetry scheduler dialog. This class implements the CcddSchedulerInputInterface class.
CcddUndoHandler.java	Class that manages GUI component undo and redo edit operations.
CcddUndoManager.java	Class that handles undo and redo of edit operations and the edit stack.
CcddUtilities.java	Class containing common utility methods used by other CCDD classes.
CcddVariableDialog.java	Dialog for the user to view the project's variables (including the each variable's full path), and to display the variables and paths using user-specified separator characters for the data type and variable names, and each variable data type/name pair. The dialog is built on the CcddDialogHandler class.
CcddVariableHandler.java	Class for building a list of project variables and converting the variable paths to unique path names, and for calculating the variable offsets. This is used for

	populating the the **Variable reference** input type selection item list and by the script data access methods.
CcddWebDataAccessHandler.java	Class that accepts web access commands and provides JSON formatted output of the requested project data.
CcddWebServer.java	Class that handles set up and management of the embedded Jetty web server.
CcddXTCEHandler.java	Class for handling import and export of data tables in XTCE XML format. This class implements the CcddImportExportInterface class.
Images.java	Dummy class required for the images folder contents to be accessible.

Appendix E.4. PostgreSQL tables

Data tables created by the user have the columns defined in the table's type definition. In addition, each data table has two initial columns that do not appear in the data table when it is edited within the application. These two columns represent the primary key (column name _key_) and the row index (column name _index_). The primary key column contains a unique, positive, sequential integer value automatically assigned by the database to each row. This value is used by the application to select specific rows in the table for modification and deletion. The row index column contains a unique, positive, sequential integer value assigned by the CCDD application. The database does not guarantee a particular order to the rows of data stored for a table; i.e., when the table's data is retrieved the row order may not be the same as the order displayed in the table editor when the data was stored. To overcome this, when a data table is loaded from the database its row index values are used to restore the row order to that specified by the user using the table editor.

In addition to the tables created by the user for containing the project's data, CCDD uses a number of internal tables for keeping track of certain information. These tables are denoted by the prefix '__' (two underscores) and do not show up in the table trees. The tables, with their descriptions and formats, are described below:

Table name:	__app_scheduler	
Description:	Contains the information produced by the application scheduler	
Columns:	time_slot	Time slot to which the application belongs in the format <*Time Slot #*>, where # is the time slot index
	application_info	Application information for the specified time slot. The information is composed of the application name, rate (in Hertz), maximum allotted run time (in seconds), priority, application wake-up ID (in hexadecimal), application wake-up name, application housekeeping send rate, housekeeping application wake-up name, housekeeping application wake-up ID (in hexadecimal), and scheduler group, separated by commas
Comment:	Application parameter values: maximum number of slots per message, maximum messages per second, maximum messages per cycle, maximum number of commands	

Table name: __associations

Description: Contains the script file and data table associations

Columns:		
	description	Description of the association; may be blank
	script_file	Script file path and file name
	member_tables	This column contains the name(s) of the table(s) associated with the script file name. If multiple tables are associated then each is separated by a semi-colon and line feed character. The table names are in the format <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]]. Child tables of an associated table are automatically included when loading the data for script execution

Comment: Unused

Table name: __data_types

Description: Contains the information for the data type definitions. The table is automatically populated by default data types, which can be altered or deleted

Columns:		
	user_name	User-defined data type name
	c_name	C-language data type name
	size	Data type size in bytes
	base_type	Base data type (signed integer, unsigned integer, floating point, character, or other)
	index	OID value; used to uniquely identify a row in the table

Comment: Unused

Table name: __fields

Description: Contains the definitions and values for all of the project's data fields. Each row in the table describes a single data field. The order that the data fields appear in this table is the same as the order of the fields when displayed with a data table, group, or project

Columns:

owner_name	Path (applicable for structure table instances) and name of the table to which this data field belongs. This column contains the parent and path to the table belonging to the group, separated by commas. This is in the format <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]]. Default data fields (i.e., those applied to each table of a given table type when created) are denoted by having an owner name in the format *Type*:<*table type name*>. Group data fields (i.e., those assigned to a group of data tables) are denoted by having an owner name in the format *Group*:<*group name*>. Project data fields (i.e., those assigned at the project level) are denoted by having an owner name *Project:*.
field_name	Field name. This is the text displayed beside the input text field
field_description	Description of the field. The description is used as the tool tip text when the mouse pointer hovers over the data field
field_size	Width of the input text field in characters. Due to character width variations when using variable-spaced fonts the actual character width can be larger than this value
field_type	Determines the allowable values that can be input into the data field. The field types are Text, Integer, Positive integer, Non-negative integer, Float, Hexadecimal, Break, and Separator
field_required	true if the data field requires a value; false if the field may be left empty. The application does not enforce entering a value into a required field, but simply uses this designation to highlight the fields that have this flag set
field_applicability	Determines, when creating tables of this type, if the data field is added: 'All tables' if the data field should be added when creating any table of this type; 'Parents only' if the field is only added to parent tables; 'Children only' if the field is only added to child tables (only applicable for structure table types)
field_value	Data entered by the user into the data field's text input field. Leading and trailing white space characters are automatically stripped off by the application before storing the value

Comment: Unused

Table name: __groups

Description: Contains the information for the user-defined data table groups

Columns:

group_name	Group name
member_tables	The first row for a group contains the group's description, prefixed by a number and a comma. The number is non-zero if the group represents a CFS application. The description is used as the tool tip text when the mouse pointer hovers over the group name in a table tree. For subsequent rows with the same group name this column contains the parent and path to the table belonging to the group, separated by commas. This is in the format <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]]

Comment: Unused

Table name: __links

Description: Contains the information for the user-defined variable linkages

Columns:

rate_name	Name of the rate column from which the rate for the variables in this link are taken
link_name	Link name
member_variables	The first row for a link contains the link's rate, in samples per second, and description, separated by a comma. The description is used as the tool tip text when the mouse pointer hovers over the link name in the link tree. For subsequent rows with the same link name this column contains the parent, table path, and variable belonging to the link, separated by commas. This is in the format <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]],<*data type*>.<*variable name*>

Comment: Unused

Table name: __macros

Description: Contains the information for the macro definitions

Columns:

macro_name	Macro name
value	Macro value

Comment: Unused

Table name: __orders

Description: Contains the information for the table column orders, based on user

Columns:		
	user_name	User name for which this column ordering applies
	table_path	Path to a table in the format: <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]] *root table name* is the name of the top-level table. For a non-structure table or a top-level structure table this is the entire table path. For a structure table that is a child of another table the path contains the top-level structure table (root table name) followed by structure name and variable name pairs leading to the target child table, separated by commas
	column_order	Contains the column numbers, as defined in the __types table, separated by colons (:), in the order in which the columns are displayed when the user, *user_name*, is viewing the table specified by *table_path*

Comment: Unused

Table name: __reserved_msg_ids

Description: Contains the reserved message IDs and ID ranges with their descriptions. By default the range 0x0800 - 0x08FF is reserved for cFE telemetry IDs and the range 0x1800 - 0x18FF is reserved for cFE command IDs (these default values may be altered or deleted)

Columns:		
	msg_id	Message ID or ID range in hexadecimal format. ID range values are separated by a hyphen (-)
	description	User-defined text describing the reserved ID or ID range

Comment: Unused

Table name: __script_<*script name*>

Description: Contains the contents of the script file <*script name*>

Columns:		
	line_number	Script file line number
	line_text	Line of text from the script file

Comment: script name with capitalization intact, script description

Table name: __table_types

Description: Contains the table type definitions for the project's data tables. Structure and command table types are created by default (these can be altered or deleted)

Columns:

type	Table type name
index	Sequential index, starting with 0, that dictates the order in which the columns appear in a table of this type. Column order can subsequently be changed by the user
column_name	Column name as used in the database. This version of the column name has the capitalization removed and spaces replaced with underscores (_)
column_name_user	Column name as seen by the user. This version of the name preserves the capitalization and spaces that the user specified when defining the column name, and is used as the column name in the data table
column_description	Description of the column. Used as the tool tip text when the mouse pointer hovers over a table's column header
input_type	Name of the column's input data type (e.g., Positive integer, Enumeration). The input data type determines what values may be entered into then column
row_value_unique	't' (true) if the value in this column cannot match the value in any other rows of this column; 'f' (false) if the value is allowed to be duplicated in other rows of this column
column_required	't' (true) if the column requires a value; 'f' (false) if the column may be left empty. The application does not enforce entering a value into a required column, but simply uses this designation to highlight the columns that have this flag set
allow_structure	't' (true) is this column allows inputs when the data type column for this row contains a structure table name; 'f' (false) if this column is to be grayed out and not allow input when the data type column for this row contains a structure name. If no data type column (a column with the input type of Primitive & Structure) is present in this table type definition then this column is ignored
allow_pointer	't' (true) is this column allows inputs when the data type column for this row contains a pointer; 'f' (false) if this column is to be grayed out and not allow input when the data type column for this row contains a pointer. If no data type column (a column with the input type of Primitive & Structure) is present in this table type definition then this column is ignored

Comment: Unused

Table name: __temp_table

Description: Temporary table created by the CCDD PostgreSQL functions

Columns: Varies

Comment: Unused

Table name: __tlm_scheduler

Description: Contains the information produced by the telemetry scheduler for the telemetry messages

Columns:

rate_name	Rate name
message_name	Message name in the format <*Message #.#*> where the first number is the message index and the second is the sub-index for the message
message_id	Message ID number, in hexadecimal
member_variable	Contains the variable's rate (in hertz) followed by a backslash (\), then the parent, table path, and variable belonging to the message, separated by commas. This is in the format <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]],<*data type*>.<*variable name*>

Comment: Rate parameters: maximum seconds per message, maximum messages per second, include uneven rates (true or false), rate 1 column name, rate 1 data stream name, rate 1 maximum messages per cycle, rate 1 maximum bytes per second [, …, rate *n* column name, rate *n* data stream name, rate *n* maximum messages per cycle, rate *n* maximum bytes per second]

Table name: __users

Description: Contains the access level assignment for eac user associated with the project

Columns:

user_name	User name. This is one of the users defined in the PostgreSQL server
access_level	User access level: 'Admin', 'Read/Write', or 'Read Only'

Comment: Unused

Table name: __values

Description: Contains the description and individual data table cell values for all of the project's data tables

Columns:

table_path	Path to a table in the format: <*root table name*>,<*level 1 child table's prototype name*>.<*level 1 child table's variable name*>[,<*level 2 child table's prototype name*>.<*level 2 child table's variable name*>[,*level 3, etc.*]] *root table name* is the top-level table name. For a non-structure table or a top-level structure table this is the entire table path. For a structure table that is a child of another table the path contains the top-level structure table (root table name) followed by structure name and variable name pairs leading to the target child table, separated by commas
column_name	Column name as seen by the user (versus the database version, which is all lower case and has any spaces replaced with undescores (_))
value	If the column_name column is empty then this column contains the table description. If the column name is not empty this column contains the value entered by the user into the specified table and column cell. Leading and trailing white space characters are automatically stripped off by the application before storing the value

Comment: Unused

Table name: *prototype table name*

Description: Contains the cell values for the table

Columns: Table type dependent

Comment: Table name with capitalization intact, table type name

Appendix E.5. PostgreSQL Functions

The CCDD application creates a number of special functions within the project's database to optimize certain table searches and updates. A description of these functions is provided below. The input(s) and output(s) are also described; the 'type' (shown in parentheses) is the PostgreSQL data type.

Function name: delete_function

Description: Deletes a function from the project database

Input (type): function_name (text) Name of the function to delete

Output (type): none

Function name:	find_columns_by_name	
Description:	Search the data tables in the specified column for non-empty cells. For structure tables include child table references in the search (i.e., references in the __values table).	
Input (type):	column_name_user (text)	Name of the column as seen by the user
	column_name_db (text)	Name of the column as used in the database
	table_types (text[]	Name(s) of the table types to which the search is constrained
Output (type):	owner_name (text)	Name of the table containing the search column name
	column_value (text)	Value of the table's cell in the search column

Function name:	find_prototype_columns_by_name	
Description:	Search the prototype data tables in the specified column for non-empty cells.	
Input (type):	column_name_db (text)	Name of the column as used in the database
	table_types (text[])	Name(s) of the table types to which the search is constrained
Output (type):	owner_name (text)	Name of the table containing the search column name
	column_value (text)	Value of the table's cell in the search column

Function name:	get_def_columns_by_index	
Description:	Search a structure table for the values in the default columns (as used in the database) that define the table as a structure. These columns are data_type, variable_name, bit_length, all rate columns, and all enumeration columns. Return the references sorted by the _index_ column, which defines the order of the variables relative to each other.	
Input (type):	name (text)	Name of the prototype structure table
Output (type):	data_type (text)	Data type (structure reference or primitive data type)
	variable_name (text)	Variable name
	bit_length (text)	Bit length; blank if the variable has no bit length assigned
	rate (text)	Rate value for each rate column defined in the table's type, in the order the rate columns are assigned in the type definition, separated by commas. The rate value is blank if the variable does not have a value assigned for a given rate
	enumeration (text)	Enumeration value for each enumeration column defined in the table's type, in the order the enumeration columns are assigned in the type definition, separated by backslashes (\). The enumeration value is blank if the variable does not have a value assigned for a given enumeration

Function name:	get_def_columns_by_name	
Description:	Search a structure table for the values in the default columns (as used in the database) that define the table as a structure. These columns are data_type, variable_name, bit_length, all rate columns, and all enumeration columns. Return the references sorted alphabetically by the variable_name column.	
Input (type):	name (text)	Name of the prototype structure table
Output (type):	data_type (text)	Data type (structure reference or primitive data type)
	variable_name (text)	Variable name
	bit_length (text)	Bit length; blank if the variable has no bit length assigned
	rate (text)	Rate value for each rate column defined in the table's type, in the order the rate columns are assigned in the type definition, separated by commas. The rate value is blank if the variable does not have a value assigned for a given rate
	enumeration (text)	Enumeration value for each enumeration column defined in the table's type, in the order the enumeration columns are assigned in the type definition, separated by backslashes (\). The enumeration value is blank if the variable does not have a value assigned for a given enumeration
Function name:	get_table_members_by_index	
Description:	Get the table member information for all prototype structure tables, sorted by the table's _index_ column, which defines the order of the variables relative to each other. The member information is used primarily to construct the table trees, so it includes the data types (both for structure references and primitives) and variable names for each prototype structure table. Also included are each variable's bit length (if any), rate values (for each rate column in the structure's type definition), and enumerations (for each enumeration column in the structure's type definition).	
Input (type):	none	
Output (type):	tbl_name (text)	Prototype structure table name
	data_type (text)	Data type (structure reference or primitive data type)
	variable_name (text)	Variable name
	bit_length (text)	Bit length; blank if the variable has no bit length assigned
	rate (text)	Rate value for each rate column defined in the table's type, in the order the rate columns are assigned in the type definition, separated by commas. The rate value is blank if the variable does not have a value assigned for a given rate
	enumeration (text)	Enumeration value for each enumeration column defined in the table's type, in the order the enumeration columns are assigned in the type definition, separated by backslashes (\). The enumeration value is blank if the variable does not have a value assigned for a given enumeration

Function name:	get_table_members_by_name	
Description:	Get the table member information for all prototype structure tables, sorted alphabetically by the variable_name column. The member information is used primarily to construct the table trees, so it includes the data types (both for structure references and primitives) and variable names for each prototype structure table. Also included are each variable's bit length (if any), rate values (for each rate column in the structure's type definition), and enumerations (for each enumeration column in the structure's type definition).	
Input (type):	none	
Output (type):	tbl_name (text)	Prototype structure table name
	data_type (text)	Data type (structure reference or primitive data type)
	variable_name (text)	Variable name
	bit_length (text)	Bit length; blank if the variable has no bit length assigned
	rate (text)	Rate value for each rate column defined in the table's type, in the order the rate columns are assigned in the type definition, separated by commas. The rate value is blank if the variable does not have a value assigned for a given rate
	enumeration (text)	Enumeration value for each enumeration column defined in the table's type, in the order the enumeration columns are assigned in the type definition, separated by backslashes (\). The enumeration value is blank if the variable does not have a value assigned for a given enumeration

Function name:	reset_link_rate
Description:	Set the rate to 0 for any links in the __links table for links containing no member variables.
Input (type):	none
Output (type):	none

Function name: search_tables

Description: Search the database tables for the specified text string. The search may be modified to ignore case, to use a regular expression in the search string, to search only the data tables (as opposed to including the internal tables), and to limit searching to specific columns in the data tables.

Input (type):	search_text (text)	Text to search for in the tables. Interpreted as a literal string unless the *allow_regex* flag is true, in which case the search text is considered a regular expression
	no_case (boolean)	true to ignore case when determining matching text; false to preserve the text case
	allow_regex (boolean)	true to interpret the search string as a regular expression; false to treat the search string as literal text
	selected_tables (text)	'ALL' to search all tables (data, internal, and script), 'DATA' to limit the search to data tables and entries in the custom values table, 'PROTO' to limit the search to the data tables only (i.e., prototype tables), 'INPUT' to search only the tables containing an input type reference (the internal table type and fields tables), or 'SCRIPT' to search only script files
	columns (name[])	one or more column names, separated by commas, to which the match is limited. Blank to include any column
	all_schema (name[])	Database schema(s) in which to limit the search. Defaults to 'public'
Output (type):	search_result (text)	Information on each match found. The text consists of multiple parts separated by the backslash (\) character:
	schema_name	Schema in which the match is found
	table_name	Name of the table
	column_name	Name of the column in the table
	table_description	Comment text for this table which includes the table's name with case preserved and the table's type, separated by a comma
	column_value	Complete contents of the table cell where the match occurs

Function name: update_data_type_names

Description: Replace all references to a structure table found in every table containing a column with the default database name for the data type ('data_type') with the new structure table name.

Input (type):	oldtype (text)	Name of the structure data type to replace
	newtype (text)	New name for the structure data type
Output (type):	none	

Appendix E.6. Known Issues

1. Concurrent operation is not supported. Simultaneously interacting with the same project from more than one instance of the CCDD application or via another database access application can result in unexpected results or corruption of the project database.

2. If the user lacks administrator privileges then when the program starts in Windows a message similar to the following may be displayed at the command prompt:

   ```
   Sep 10, 2014 3:06:17 PM java.util.prefs.WindowsPreferences <init>

   WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at
   root 0x80000002.  Windows RegCreateKeyEx(...) returned error code 5.
   ```

   This is a result of Windows attempting to create a global registry entry for the program preferences, even though only a user entry is requested. The user entry is successfully created/updated, so the warning message may be ignored. The message can be eliminated by executing the application once as an administrator since this adds the missing key. Adding the `Prefs` key manually is also an option.

3. When using the GTK+ look and feel in Linux, or any look and feel in Windows, the Files selection box does not highlight the files initially selected when the file choosing dialog is opened. The file name list does reflect the currently selected files, however.

4. The HTML-formatted table columns cease character wrapping below a certain width (which is dependent on the cell contents), causing the text to appear truncated. Character and word wrapping work as expected when the column width exceeds this 'limit'.

5. During some operations (for example, exporting tables) a large number of Command events are generated and logged. The GUI is locked during this period. Depending on the number of events generated, disabling display of the Command events, via the check box at the bottom of the main window or command line option, can significantly lessen the duration that the GUI is locked.

6. If 32-bit Java 7 is used in a 64-bit Linux environment then the 32-bit compatibility libraries must be installed. The specific libraries are Linux version dependent. As an example, the user's guide cannot be displayed in 64-bit CentOS 6 using the command menu unless the Gnome 32-bit library, libgnome.i686, is installed.

7. In Java 9 and subsequent versions the JAXB libraries are no longer part of the default Java installation. For Java 9 and 10, in order for these libraries to be accessed the option `--add-modules java.xml.bind` must be added to the CCDD startup command. This 'fix' will no longer be valid beginning with Java 11.

8. When using certain Microsoft wireless mice running under Microsoft Windows the mouse wheel rotation is misinterpreted in Java applications. The issue has to do with the higher resolution capabilities of these mice. To allow this type of mouse to work properly with Java perform the following steps (note that if the scrolling problem returns following a reboot, then uninstall the mouse and mouse drivers and redetect the mouse - in Device Manager the mouse description should show as "HID-compliant mouse"; the steps below can then be performed):
   a. **Control Panel** → **Mouse**
   b. **Mouse Properties** → **Hardware** tab
   c. Select the problematic mouse from the list ("HID-compliant mouse")
   d. Click the **Properties** button
   e. Go to the **Details** tab

f. Select "Device Instance Path" from the combo box
g. A value will be displayed (e.g.: HID\VID_045E&PID_0745&MI_01&COL01\8&5538EC&0&0000); note this value. This is the path of the registry key that corresponds to this instance of the mouse
h. Open the registry editor and navigate to:

   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\<*value noted in step 7*>\Device Parameters

i. In Device Parameters, add the following DWORD (32 bit) registry keys:

   HScrollHighResolutionDisable = 1

   VScrollHighResolutionDisable = 1

   Delta = 120 (decimal)

j. Unplug, then plug back in the mouse transceiver to re-initialize the driver
k. The wheel scrolling should work in Java after this. If the scroll speed is too fast then perform the remaining steps
l. **Control Panel** → **Mouse**
m. **Mouse Properties** → **Wheel**
n. Under **Vertical Scrolling** set "Roll the wheel one notch to scroll: The following number of lines at a time:" to 1
o. Select the **OK** button
p. Open the **Mouse and Keyboard Center**
q. Under **Basic Settings** select **Wheel**
r. Adjust the **Wheel Vertical Scrolling** slider to the slowest setting