

Core Flight System Command and Data Dictionary Utility Tutorial

Engineering Directorate
Software, Robotics, and Simulation Division

Version 1.2
September 2018



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



Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: <i>September 2018</i>	Page 2 of 48

Contents

1.0	Description	5
2.0	Requirements.....	5
3.0	Tutorial.....	5
3.1	Start CCDD.....	5
3.2	Create a project	6
3.3	Create a data table.....	7
3.4	Open a data table for editing.....	7
3.5	Edit a data table	8
3.5.1	Add a variable	8
3.5.2	Create an array variable.....	10
3.5.3	Rearrange the rows.....	11
3.5.4	Rearrange the column order.....	11
3.5.5	Enumerations.....	11
3.5.6	Padding for byte alignment.....	12
3.5.7	Data fields	13
3.6	Edit a table type	16
3.6.1	Table type data fields.....	18
3.7	Create a child table	19
3.8	Grouping tables.....	21
3.8.1	Group data fields.....	24
3.8.2	CFS applications	25
3.9	Create a script association	25
3.10	Execute a script association	28
3.11	Data types	29
3.12	Macros	31
3.13	Message IDs	33
3.14	Commands	34
3.14.1	Add a command argument to the Command table type.....	34
3.14.2	Create a command table and add command information	36
3.14.3	Execute a script using the command information	38
3.15	Importing and exporting tables	40
3.15.1	Create an import file.....	40
3.15.2	Import the table	41
3.15.3	Export the table	43
3.16	Telemetry scheduling.....	45
3.16.1	Telemetry rates.....	45
3.16.2	Assigning rates to variables.....	47
3.16.3	Linking variables.....	47
3.16.4	Bit length and bit-packing	47
3.16.5	Creating the housekeeping copy table	48

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 3 of 48

Figures

Figure 1.	CCDD main application window.....	6
Figure 2.	Create Project dialog.....	6
Figure 3.	New Table dialog.....	7
Figure 4.	Edit Table dialog.....	8
Figure 5.	Table editor	9
Figure 6.	Empty row inserted into the data table.....	9
Figure 7.	Variable name and data type.....	10
Figure 8.	Array definition and array member rows	11
Figure 9.	Variable “myArray1” moved above “myVar1”	11
Figure 10.	Enumerated variable.....	12
Figure 11.	Add/update/remove Padding dialog	13
Figure 12.	Padding applied to align the variables	13
Figure 13.	Data field editor	14
Figure 14.	Data field editor with row inserted.....	14
Figure 15.	Data field editor with fields defined	15
Figure 16.	Table with data fields added	15
Figure 17.	Table type editor	16
Figure 18.	Default structure table type.....	17
Figure 19.	New structure table type column definition.....	18
Figure 20.	Structure table “myStruct” with the new “Lower limit” column.....	18
Figure 21.	New column with values entered	18
Figure 22.	Table type with default field assigned	19
Figure 23.	Variable added to structure “myChildStruct”	19
Figure 24.	Child structure variable added to structure “myStruct”	20
Figure 25.	A child table and its prototype.....	20
Figure 26.	Overriding inheritance of a table cell.....	21
Figure 27.	Restoring a cell’s inheritance	21
Figure 28.	Group manager	22
Figure 29.	Assigning a table to a group.....	22
Figure 30.	Multiple groups	23
Figure 31.	Table tree, filtered by groups.....	23
Figure 32.	Group data field	24
Figure 33.	Group with data field assigned	24
Figure 34.	Script manager	25
Figure 35.	Script selection dialog	26
Figure 36.	Script associated with a table	27
Figure 37.	Script associated with a group.....	28
Figure 38.	Script output file	29
Figure 39.	Data type editor	30
Figure 40.	New data type	30
Figure 41.	Added data type.....	31
Figure 42.	Macro editor	31
Figure 43.	Added macros	32
Figure 44.	Macro combo list	32
Figure 45.	Macro combo list; filtered.....	32

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: <i>September 2018</i>	Page 4 of 48

Figure 46.	Array size change via macro update	32
Figure 47.	Assign message IDs dialog.....	33
Figure 48.	Reserve ID editor.....	34
Figure 49.	Command table type.....	35
Figure 50.	Command table type with second command argument columns.....	36
Figure 51.	Command information added to table “myCommand”	37
Figure 52.	Script output file with commands.....	39
Figure 53.	Table definition in a spreadsheet.....	40
Figure 54.	Spreadsheet saved in CSV format.....	41
Figure 55.	Import Table(s) dialog.....	42
Figure 56.	Imported tables in the table editor.....	43
Figure 57.	Export table(s) dialog	44
Figure 58.	Table exported in CSV format	45
Figure 59.	Rate Parameters dialog.....	46
Figure 60.	Rate Parameters dialog with values entered.....	47

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 5 of 48

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. See JSC-37494, Core Flight System Command and Data Dictionary Utility User's Guide, for details on the CCDD application. This tutorial provides an example of setting up a project using CCDD, populating the project's data tables, and using the project data to create output files.

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 must be installed per the user's guide, including access to a PostgreSQL server. The user must have a valid login role on the PostgreSQL server. This tutorial is valid for CCDD version 1.4.22 and subsequent versions.

3.0 Tutorial

The following paragraphs lead the user through the basics of using the CCDD application, beginning with creating a new project, adding data tables to the project, entering data, and using a script to generate an output product from the project's data. The paragraphs follow a specific order, with successive ones building on previous ones. The results may not be the ones expected if the steps are executed out of order.

3.1 Start CCDD

Start the CCDD application. The main application window appears (see Figure 1). A login dialog may also appear depending on the postgres server's authentication settings; enter your postgres user login and password. In the **Project** column "**server**" is shown once a connection to the postgres server is established, or "**none**" if the connection to the server doesn't exist. The server must be connected in order to proceed with the tutorial. The **User** column indicates the name of the user that is connected.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 6 of 48

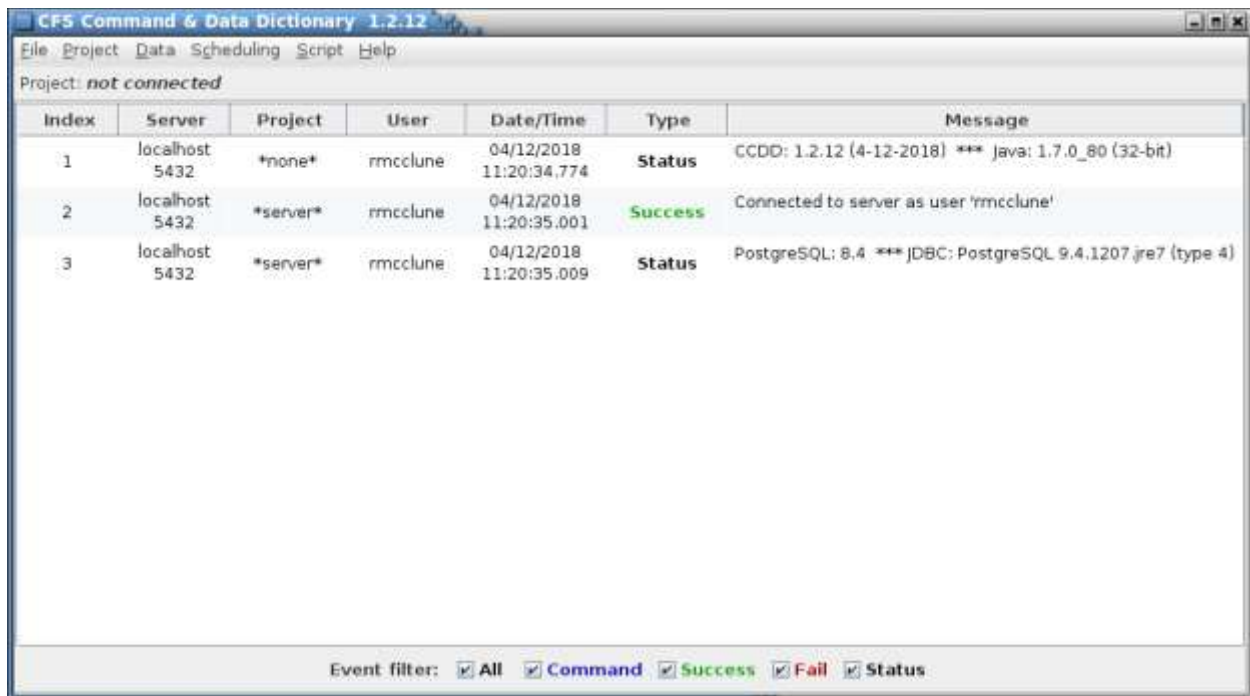


Figure 1. CCDD main application window

3.2 Create a project

In the main window's menu bar select **Project | New**. A dialog appears similar to that shown in Figure 2 with the postgres and your user roles under the **Select project owner** heading (more roles may also be present if extant on the postgres server).

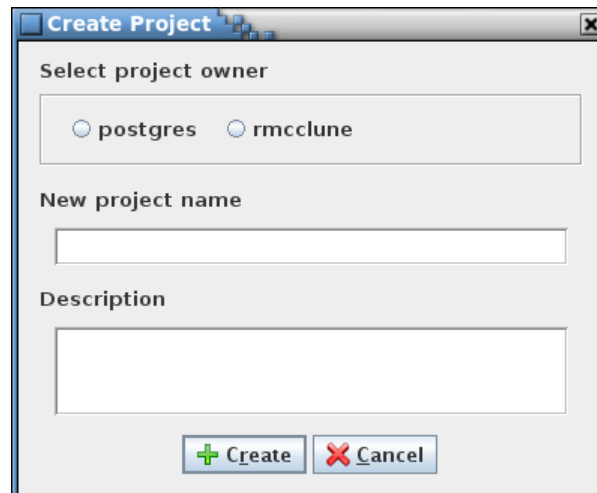


Figure 2. Create Project dialog

Select the radio button for your user as the project owner. Only the owner is allowed to make changes to the data stored in the project. It's possible to assign a generic owner and then link the users to this generic owner, allowing multiple users to make changes. In the **New project name** input field type "MyProject" as the project name. The project name has constraints as to characters allowed, length, and uniqueness. Enter a description for the project in the **Description** field (this can be altered later if

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 7 of 48

desired). Press the **Create** button to create the new project. The event log displays a message when the project is successfully created (or if an error occurs).

Now that the project exists data can be added to it. Data is stored as tables and table entries in the postgres database. The project isn't entirely empty initially; aside from a number of necessary internal tables and postgres functions, the table types, data types, and reserved message IDs are populated with default information. CCDD handles all of the transactions with the database, reading and writing data to the tables as needed. These transactions are reflected in the main window's event log entries.

A project must be opened before in order to alter the data within it. In the main window's menu bar select **Project | Open**. The **Open project** dialog that appears displays all of the CCDD projects for which the current user has access. Select "MyProject" from the dialog and press the dialog's **Open** button. The event log indicates that the project is open and locked. Locking the project prevents all but the current user from making updates to the project.

3.3 Create a data table

The primary location for a project's data are in structure and command tables. A structure table mimics information commonly found in a C language structure. In the main window's menu bar select **Data | New table(s)**; the **New table** dialog appears (Figure 3).

By default table types exist for structure and command tables. These table types may be modified or deleted, and additional types can be added. A table type serves as the template for new data tables, defining the columns and input constraints for that type of table. Select the **Structure** radio button under the **Select table type** heading. Enter "myStruct" as the table name in the **Table name(s)** field. A table name must begin with a letter or underscore, and can contain letters, numerals, and underscores. In the **Description** field enter "My structure table". Figure 3 shows the completed dialog. Press the **Create** button to create the table 'myStruct'.

Figure 3. New Table dialog

3.4 Open a data table for editing

Now that a table exists in the project data can be entered into it. The first table created represents a structure, so the data to be entered will be variable names, data types, etc. In the main window's menu

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 8 of 48

bar select **Data | Edit table(s)**, which causes the **Edit Table** dialog to appear (see Figure 4). Data tables are displayed using a tree format (similar to folders and files on a computer). The tables are arranged under two headings, **Prototypes** and **Parents & Children**. The tree is initially collapsed, so only these two headings are displayed. Select the **Expand all** check box below the tree, as shown in the figure, to display the tables under each heading.

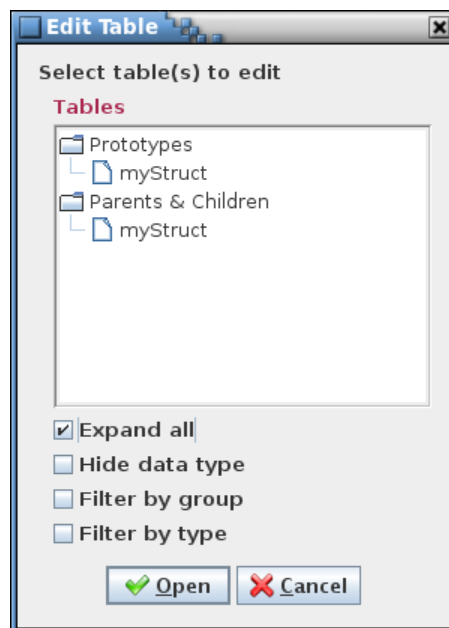


Figure 4. Edit Table dialog

All tables created using the **Data | New table(s)** command appear under the **Prototype** heading. A prototype can be thought of as a “rubber stamp” from which copies of the table are made. The copies are created when a table becomes the “child” of another table; that is, the child is referenced by the parent table. In practice this only occurs with structure tables since a structure can have a variable with another structure as the data type. The tables under the **Parents & Children** heading reflect this relationship. Any table that isn’t referenced by another table appears in the first level under the **Parents & Children** heading; these are referred to as “root” and “top-level” tables in the user’s guide. Child tables appear on subsequent “nest” levels. This nesting can extend to any depth, dependent on the structure references.

Use the mouse or keyboard to select the table “myStruct” from either heading (these represent the same table since “myStruct” is both a prototype and a root table). Press the **Open** button to open the table in a table editor.

3.5 Edit a data table

3.5.1 Add a variable

In the previous section the table “myStruct” was selected from the table tree in the **Edit table(s)** dialog for editing. The table editor, shown in Figure 5, appears. The editor can be resized if needed. The editor consists of a menu bar (along the top), the table (initially empty), the description field, a data field, and a number of buttons. A tab above the table displays the table’s name (if multiple table are open then a tab appears for each table). Notice that the table’s description, entered when the table was created, appears in the **Description** field.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 9 of 48

The buttons provide a means of accessing some of the more commonly used editor commands; these (and other) commands can also be found in the editor's menu bar. CCDD has a number of editor and manager dialogs for altering the project's data. A common attribute of these editors and managers is that no data is actually stored in the project's database until the user explicitly chooses to do so. As long as there are unstored changes an asterisk appears beside the table's name in the tab. Also, if the table is closed and there are unstored changes a dialog appears allowing the user to choose between continuing (and discarding the changes) or returning to the editor.

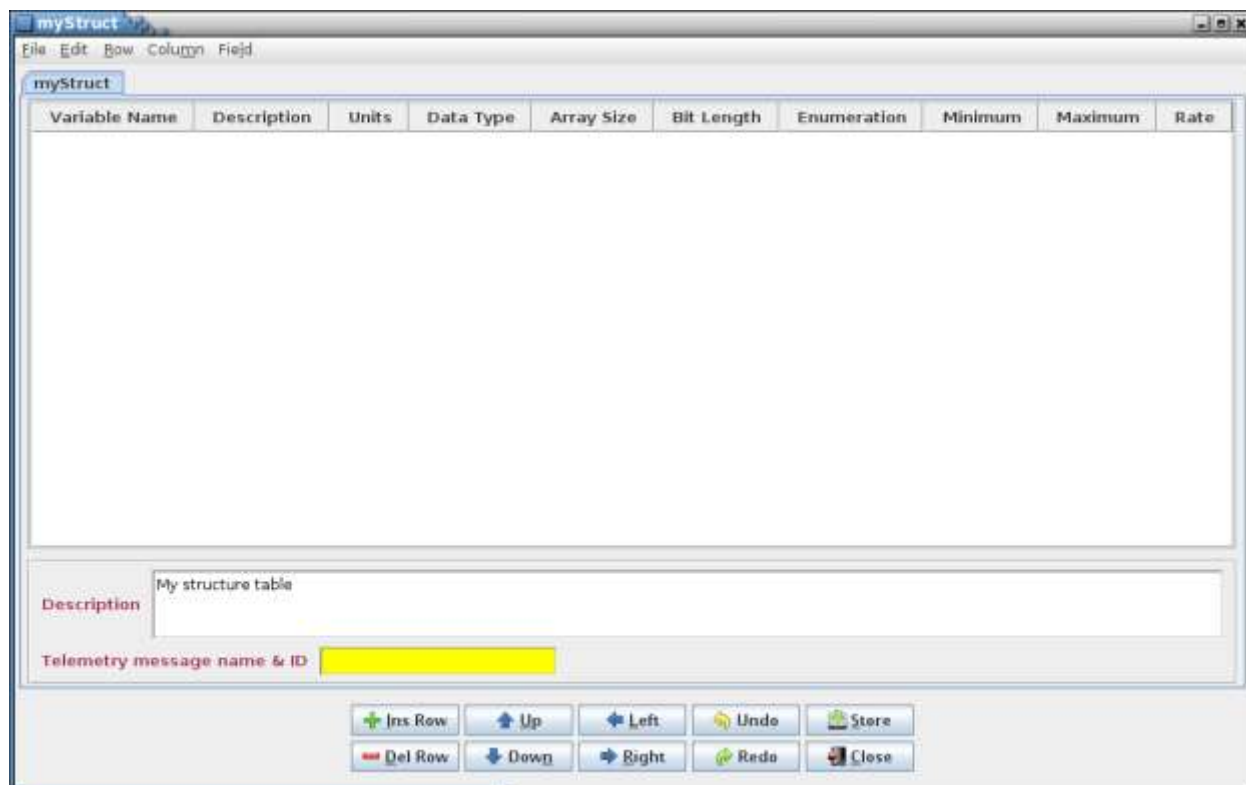


Figure 5. Table editor

A row must be inserted into the table before data can be entered, so press the **Ins Row** button (or alternatively, select **Row | Insert row** from the editor's menu bar). An empty row appears in the table as in Figure 6.

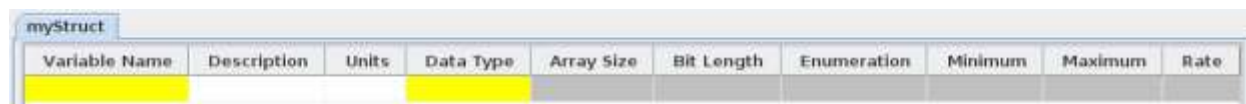


Figure 6. Empty row inserted into the data table

A cell with a gray background cannot be edited. Entering data into the other cells can determine if the grayed out cell becomes available for editing; if so, the cell's background changes to white. A cell with a yellow background indicates that a value is required in the cell. "Required" in this context means that the row needs this information at a minimum under normal circumstances and serves as a reminder to the user – in general the CCDD application doesn't enforce entering data into cells deemed as "required".

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 10 of 48

Since this table represents a structure each row is a variable definition, so a variable name and data type are required at a minimum to define the variable. In the **Variable Name** cell enter “myVar1”. Notice that the **Array Size** cell is now editable. Select the **Data Type** cell using the mouse or keyboard; a combo box appears from which a data type is chosen (Figure 7).

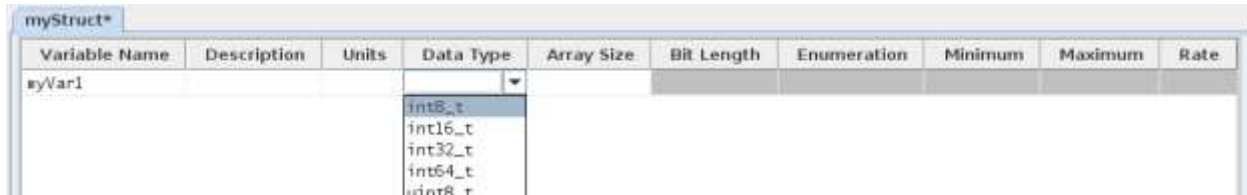


Figure 7. Variable name and data type

The choice of data type is constrained to those appearing in the list. There are two types of data type: primitive and structure. Primitive data types are those representing integers, floating points, characters, or pointers (a later section demonstrates changing the primitive data types). Structure data types appear if there are other structure tables defined – this is how child structures are created. Notice that “myStruct” doesn’t appear in the data type combo box. This is because a structure isn’t allowed to reference itself. In fact, CCDD prevents displaying any structure that would result in a circular reference, regardless of the depth of child structures involved. For this tutorial select “uint16_t”, a two byte unsigned integer as the data type for variable “myVar1”. The remaining cells that were grayed out are now editable.

Press the **Store** button to store the table, including the newly defined variable, into the project database. Press the **Okay** button in the confirmation dialog appears; the data is now stored (the event log records this action).

3.5.2 Create an array variable

Continuing from the last section, press the **Ins Row** button again to insert a second row into the table. The new row appears below the original row. New rows appear at the bottom of the table unless a row in the table is highlighted (i.e., one or more cells in the row is selected), in which case the row is inserted immediately below the highlighted row.

In the new row enter “myArray1” as the variable name and select “float” as the data type. Notice that the **Bit Length** and **Enumeration** cells are grayed out since this information isn’t applicable to a floating point value. Enter “3” into the **Array Size** cell. This creates a single dimensional array with three members. The members aren’t visible by default; change this by selecting **Row | Expand arrays** in the editor’s menu bar (or by double clicking the right mouse button while the mouse pointer is in any row in the **Array Size** column). Three more rows, one for each array member, are displayed underneath the new row, which is the array’s definition row. Notice that many of the cells for the members cannot be edited since these are dependent on the definition, whereas others, such as the **Description**, can be assigned per member. See Figure 8.

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate
myVar1			uint16_t						
myArray1			float	3					
myArray1[0]			float	3					
myArray1[1]			float	3					
myArray1[2]			float	3					

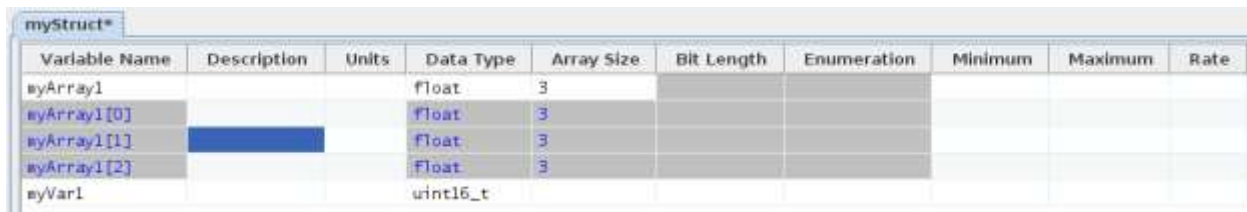
Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 11 of 48

Figure 8. Array definition and array member rows

If the array definition row is changed then the array's members are updated to reflect the change. Change the array size from "3" to "4" and note that another member row is automatically inserted with the array information in place. Change the array size back to "3" and the added member row is removed. Changing the variable name or data type for the array's definition also updates the members accordingly. The array member rows can be hidden again performing the menu command or the mouse action again. Store the changes in the project database.

3.5.3 Rearrange the rows

A variable's position within the table can be changed by selecting a cell in the row, then pressing the **Up** and **Down** buttons (or using the equivalent **Row** menu commands). Use the mouse or keyboard to highlight any cell in one of the "myArray1" rows, then press the **Up** button. The entire array, definition and member rows, is moved above the row containing the variable "myVar1" (Figure 9).



Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate
myArray1			float	3					
myArray1[0]			float	3					
myArray1[1]			float	3					
myArray1[2]			float	3					
myVar1			uint16_t						

Figure 9. Variable "myArray1" moved above "myVar1"

Rows representing more than one variable can be moved as a block. The variables must be contiguous, so any intervening variables are moved as well. Store the changes in the project database.

3.5.4 Rearrange the column order

The order in which the columns are displayed can be changed if desired. This is accomplished by one of three methods: the **Left** and **Right** buttons, the equivalent **Column** menu commands, or dragging the column using the mouse. For the button or menu method highlight a cell in the column(s) to be moved, then issue the command. Note that intervening columns are moved as well if the selected columns aren't contiguous. To use the mouse, position the mouse pointer over the column header, then press and hold the left or right mouse button. Move the mouse to reposition the column(s). Column order changes are recalled if stored in the project database. The ordering is stored by table and user; in other words, each user can order the columns in each table to their own liking and the columns appear in that order when the table is reopened.

3.5.5 Enumerations

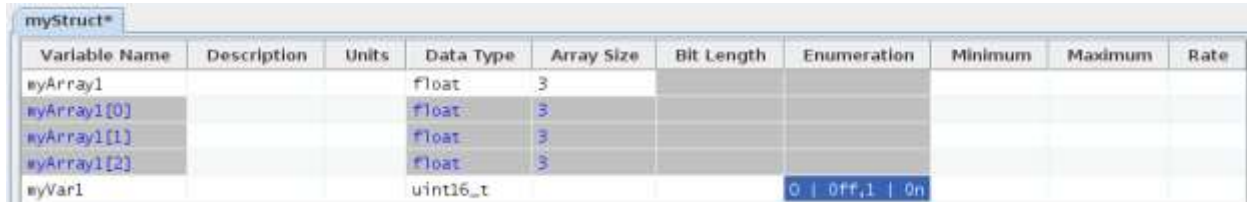
The enumeration column provides a means of defining enumerated values and their corresponding text values (and, optionally, other information). Since the enumerations are based on integer values, the **Enumeration** column is editable only when the **Data Type** column is a type of signed or unsigned integer. The format for an enumeration isn't fixed – the user is free to enter the data as desired as long as the separator characters are consistent within each enumeration. The script data access methods can then automatically detect the separation characters when parsing an enumeration.

In general the format has each enumerated value and its parameters separated from the next one by a separation character (or characters). The separation character could be a comma, for example. The individual parameters for each enumerated value are separated by another character (or characters); for example a vertical bar (|). At a minimum each enumerated value requires a text representation to

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 12 of 48

go with it. Other parameters may also be added as parameters to each enumerated value; an example would be text and background colors for use when coloring the text on a display.

For the tutorial the variable “myVar1” will be given enumeration parameters for the values 0 and 1, with the value of 0 associated with the text “Off”, and the value 1 with “On”. This is done by entering “0 | Off, 1 | On” in the **Enumeration** cell for the variable “myVar1”. Figure 10 shows the table with the change entered. Store the changes in the project database.



Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate
myArray1			float	3					
myArray1[0]			float	3					
myArray1[1]			float	3					
myArray1[2]			float	3					
myVar1			uint16_t			0 Off, 1 On			

Figure 10. Enumerated variable

DO SOMETHING WITH THE ENUMERATION IN THE TUTORIAL SCRIPT

3.5.6 Padding for byte alignment

Byte alignment refers to positioning variables on word boundaries so that CPU requests can efficiently access the variable values in memory. However, when a structure is created, the variables may not naturally align as needed. The variables must be aligned based on the size of the largest element within the structure, including any referenced child structures and their variables (and their children, etc.). For example, if the first variable is a character and the second a 2-byte integer then the second variable would not be aligned properly – it needs to be shifted one byte so that it falls on the next alignment position. Or if the second variable is a float (4 bytes), then it needs to be shifted by 3 bytes to fall on the next word boundary.

Byte alignment requirements can be met by manually inserting dummy variables so as to force the next variable to align correctly. This process is prone to error, can consume a lot of time if a large number of variables and/or tables are involved, and needs to be updated as variables are inserted, deleted, moved, have their data types, array sizes, or bit lengths changed. To eliminate these issues, the CCDD application can perform the task of adding, updating, or removing padding variables upon user command.

Select the **Data | Padding** menu command. The dialog shown in Figure 11 appears. The table tree allows selection of the prototype structure table or tables that will have the padding adjusted. Select “myStruct” from the tree and press the **Add/Update** button; the table is automatically padded as needed to align the variables on the appropriate byte boundary. Note that the addition of the padding variables doesn’t register as an unstored change for the table. This is because padding adjustments are automatically stored when the padding operation is performed.

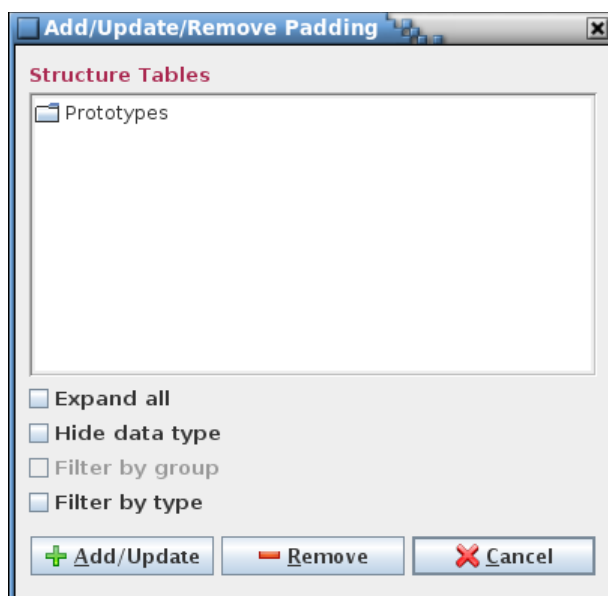


Figure 11. Add/update/remove Padding dialog

The structure table “myStruct” now appears as in Figure 12. A padding variable is added to “fill out” the structure’s size. The array “myArray1” contains three floats. Each of these occupies four bytes, so are already aligned. The next variable, “myVar1”, is also aligned since it immediately follows the last float in myArray1. The overall size of the structure, however, is two bytes deficient, so padding variables are appended – in this case as an array of two characters. Notice that the padding variable rows are highlighted for easy identification. Also notice that the padding variable name is in a specific format – this is how the application recognizes the padding variables.

myStruct									
Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate
myArray1			float	3					
myArray1[0]			float	3					
myArray1[1]			float	3					
myArray1[2]			float	3					
myVar1			uint16_t			0 Off, 1 On			
pad0__			char	2					
pad0__[0]			char	2					
pad0__[1]			char	2					

Figure 12. Padding applied to align the variables

Executing the **Padding** command and selecting **Add/Update** removes any padding variables currently in place in the table, then adds padding variables back in as needed. This should be done if variables are added, removed, or have their data type, array size, or bit length altered. Execute the menu command **Data | Padding**, choose the tables from which to remove the padding variables, and then select **Remove** to delete the selected table’s padding variables.

3.5.7 Data fields

Some data associated with a table may not be appropriate in a tabular format. For example, a value that is the same for every row, or a value that applies to the entire table (the table description falls into this category). To account for this type of information the CCDD application allows for one or more data fields to be assigned to each table. These fields appear between the table’s **Description** field and the

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 14 of 48

buttons in the table editor. By default each root structure table is assigned a data field named “Telemetry message name & ID”; this can be seen in Figure 5. Default fields can be added, modified, or removed – this is covered in paragraph 3.6.1 of this tutorial.

Select the menu command **Field | Manage fields**. The **Data Field Editor**, shown in Figure 13, is displayed.

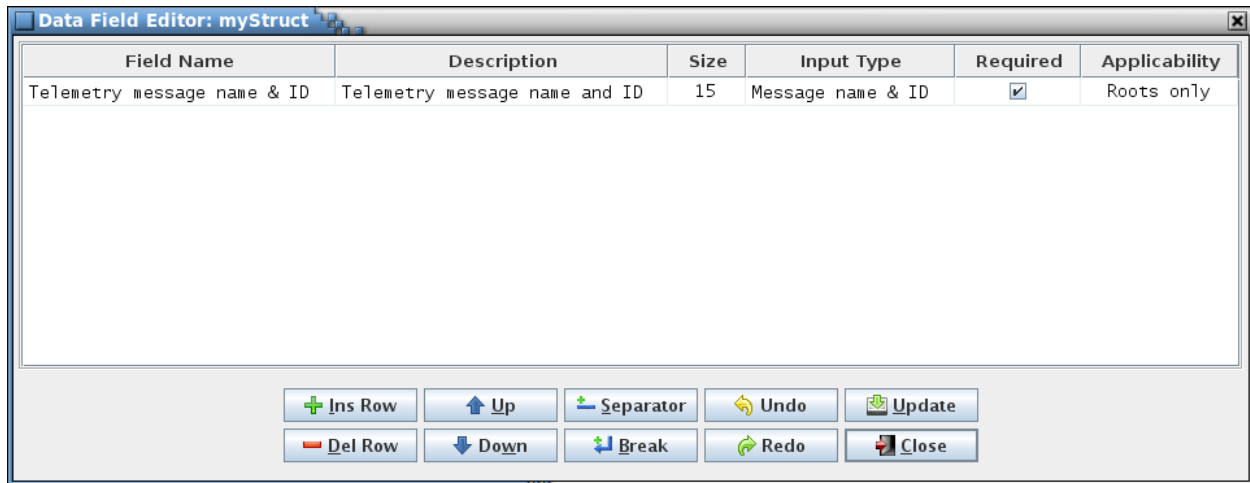


Figure 13. Data field editor

Since a default data field was assigned to “myStruct” the field editor is initially populated with this field’s definition. A data field is added by pressing the **Ins Row** button (Figure 14).

Field Name	Description	Size	Input Type	Required	Applicability
Telemetry message name & ID	Telemetry message name and ID	15	Message name & ID	<input checked="" type="checkbox"/>	Roots only
			Text	<input type="checkbox"/>	All tables

Figure 14. Data field editor with row inserted

Again, as with the table editor, a cell with a yellow background is considered required and a value must be entered for these cells. The **Field Name** is the text that appears beside the input field when it’s displayed. The **Description** cell is optional; its text appears as a tool tip when the mouse pointer is hovered over the field. The **Size** cell determines the width in characters of the field. Note that this doesn’t constrain the number of characters that can be entered into the field.

The **Input Type** cell, when selected, displays a combo box containing all of the available input types. The input type constrains the characters that can be entered into the data field once it’s created. A large number of types are provided, and custom types may be created. The default is “Text”, which allows any characters to be entered, though leading and trailing white space characters are automatically removed. Some of the data types have specific uses not associated with data fields (e.g., “Variable name” and “Array index”); these can be used to constrain the input value, but any additional meaning isn’t used by the data field. The default field, “Telemetry message and & ID”, has the input type “Message name & ID”. This imposes the constraint on the field so that only a message name (alphanumeric value) and/or message ID (hexadecimal value) can be entered in the field. This input type is recognized by the application when showing and assigning message IDs. This is discussed further in a section 3.13 of this tutorial.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 15 of 48

The **Applicability** column determines when the field is displayed in the table editor. When the column is selected a combo box appears displaying the three types of applicability: “All tables”, “Roots only”, and “Children only”. If “All tables” is selected then this field is always displayed with the table in its editor. For “Roots only” the field is displayed if the table is a root table, but not if it becomes a child table. Finally, for “Children only” the field appears in the editor if the table is a child table, but not if it’s a root table. The default field, “Telemetry message name & ID”, has the applicability set to “Roots only”, so this field appears for “myStruct” as it is a root structure table.

The **Required** cell contains a check box; if selected the field’s background color is yellow until a value is entered. This can be used to alert a user than a value is “required” for the field (though CCDD doesn’t enforce entering a value in this case).

For the tutorial an additional data field will be created. In the **Field Name** column type the field name, “Subsystem”. Use “7” for the field size. For the input type use “Alphanumeric”. Selected the **Required** check box. This field should appear whether the table is a root or child, so leave the applicability set to “All tables”. The editor should appear as in Figure 15.

Field Name	Description	Size	Input Type	Required	Applicability
Telemetry message name & ID	Telemetry message name and ID	15	Message name & ID	<input checked="" type="checkbox"/>	Roots only
Subsystem		7	Text	<input checked="" type="checkbox"/>	All tables

Figure 15. Data field editor with fields defined

Press the **Update** button. This causes the fields to be displayed into the table editor. It does not store the fields in the project’s database; this is done using the table editor’s **Store** button. Exit the data field editor by pressing its **Close** button. The table editor should now appear as in Figure 16.

Figure 16. Table with data fields added

Since the **Required** check box was selected for the “Subsystem” field and the field is empty its background is yellow. The data fields can be edited the same as is done for the table cells and **Description** field. Enter “MySystem” in the **Subsystem** field and “MY_MSG_ID q” for the **Telemetry message name & ID** and press the **Store** button. A dialog appears indicating that the field contains invalid characters, and that a message name and ID is expected. Fields and cells with the “Message

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 16 of 48

name & ID” input type expect to have a hexadecimal value (with or without the leading “0x”) for the ID portion. All table cells and data fields are checked for valid inputs based on their respective input types. Press the warning dialog’s **Okay** button. The “Message ID” field reverts to its former value, in this case empty. Leave the field empty and press the **Store** button to store the changes. Leave the table editor open for now.

3.6 Edit a table type

As demonstrated previously the table types are used as a template for creating tables. CCDD comes with two table types defined, **Structure** and **Command**. However, these types contain only the fundamental columns for these types of tables. For example, the **Command** table type has columns for only one argument, but multiple arguments may be needed. The table type editor allows altering the table type templates and creation of new ones.

Open the table type editor by selecting **Data | Manage table types** from the main window menu bar. The editor appears as shown in Figure 17.

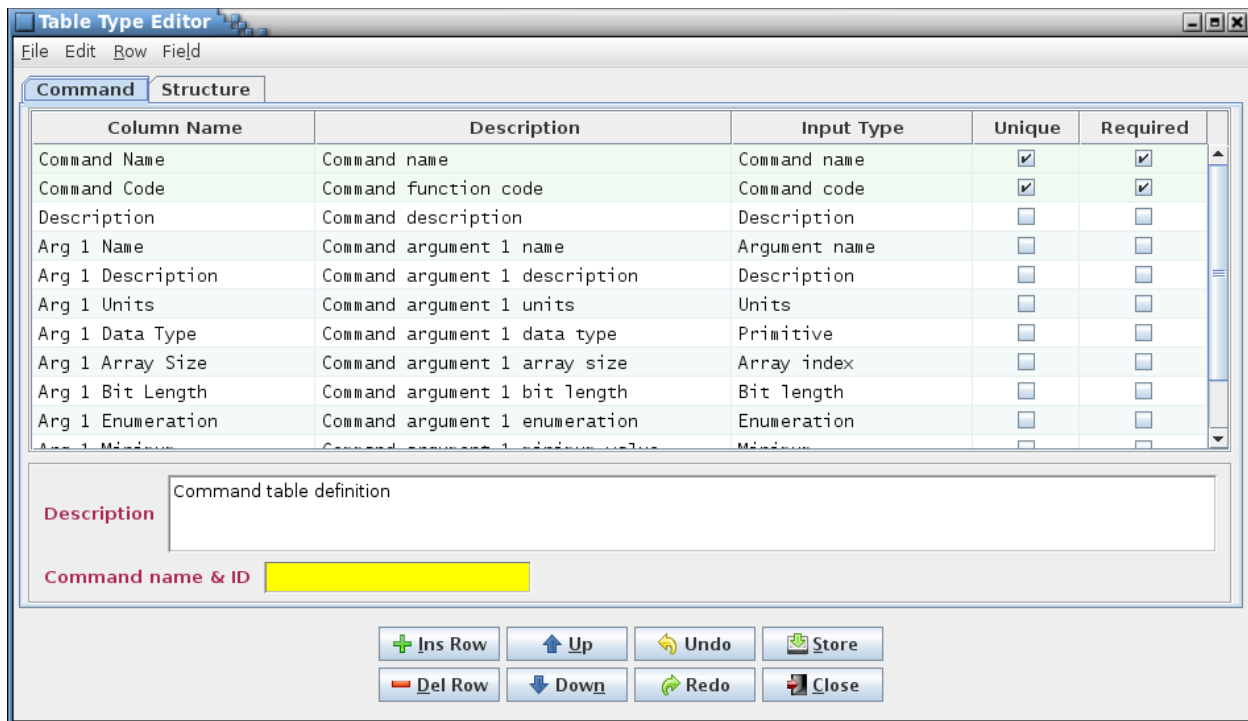


Figure 17. Table type editor

The table type editor appears very similar to the table editor, and operates in a similar manner. Each row in the table type editor table represents the definition of a column that is displayed in a table created from the table type. Select the **Structure** tab and the editor now displays the structure table type information (Figure 18).

Column Name	Description	Input Type	Unique	Required	Enable if Structure	Enable if Pointer
Variable Name	Parameter name	Variable name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Description	Parameter description	Description	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Units	Parameter units	Units	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Data Type	Parameter data type	Primitive & Structure	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Array Size	Parameter array size	Array index	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Bit Length	Parameter number of bits (bit values only)	Bit length	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Enumeration	Enumerated parameters	Enumeration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Minimum	Minimum value	Minimum	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Maximum	Maximum value	Maximum	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Rate	Downlink data rate, samples/second	Rate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Description Telemetry and data structure table definition

Telemetry message name & ID [Yellow box]

Figure 18. Default structure table type

Certain rows in the table type are highlighted (other than the standard alternating background that aids in delineating rows). These rows are considered fundamental for describing the table type (in this case a structure). For example, a structure, in order to be a structure, must contain a variable name and therefore must have a column definition for it.

Compare the rows in the structure table type to the columns in the “myStruct” table in the table editor (which should still be open). Notice that the text in the rows for the **Column Name** column match the columns names in the table editor’s column headers. The order of the column definitions determines the initial order of the columns when a table of the type is created (but recall that the order can be changed in the table editor if desired). The type editor’s **Description** column contains text that is used as tool tip text when the mouse pointer is hovered over the column’s header in the table editor.

The **Input Type** column performs like the one in the data field editor described previously. Input types are used to define and constrain the input in a table cell. Referring to Figure 18, the first row in the table type editor defines the column **Variable Name**. Its input type is “Variable name”, which constrains the value input to valid C language variable name. It also acts as a flag to other parts of the application that this value is a variable name. A structure table type may only have a single variable name column defined. The name of the column may be anything, but the input type must be “Variable name” for it to be recognized as such.

The remaining columns are described in detail in the user’s guide.

For the tutorial a new column will be added to the “Structure” table type. In the table type editor press the **Ins Row** button; an empty row appears at the bottom of the table. Enter values in the row so that it appears as shown in Figure 19.

Rate	Downlink data rate, samples/second	Rate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Lower Limit	Lower limit	Positive integer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 19. New structure table type column definition

The new column has the name “Lower Limit” and only accepts positive integer values (whole numbers > 0). The column is marked as “required”, so in the table editor its background is yellow until a value is entered into it. Press the **Store** button in the table type editor and confirm the update. Select the table editor containing “myStruct” and notice that the new column has been automatically added to the table. Resize the editor or use the horizontal scroll bar at the bottom of the table if necessary for the column to be visible. Figure 20 shows the editor’s appearance with the new column.

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myArray1			float	3						
myArray1[0]			float	3						
myArray1[1]			float	3						
myArray1[2]			float	3						
myVar1			uint16_t			0 0FF,1 0n				

Figure 20. Structure table “myStruct” with the new “Lower limit” column

Notice that each “myArray1” array member can have its own value for the new column, as can the array definition row. If a value is entered in an array’s definition row it is automatically propagated to every member. Attempt to enter values that are not positive integers (the input type assigned to the new column); a warning dialog appears when cell editing ends and the cell reverts to its former value. Enter the values “1”, “2”, “3”, and “4” for the “myArray1” array members and “myVar1” respectively – see Figure 21. Store the updates.

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myArray1			float	3						
myArray1[0]			float	3						1
myArray1[1]			float	3						2
myArray1[2]			float	3						3
myVar1			uint16_t			0 0FF,1 0n				4

Figure 21. New column with values entered

3.6.1 Table type data fields

As with tables, data fields can be assigned to a table type. However, the purpose differs; data fields for a table type are used as part of the template for tables of the type. In other words, when a table is created of the type, the table also has data fields automatically assigned to it based on the ones in the type definition. Open the table type’s data field editor using the **Field | Manage fields** menu command. The data field editor appears. This **Applicability** column only appears if the table type references a structure, so if the data field editor is opened for the **Command** table type this column does not appear.

A single default field, “Telemetry message name & ID”, already exists for the **Structure** table type. For the tutorial a second default data field, named “Subsystem” (as was added to the table “myStruct”), should be included with all root tables. Press the **Ins Row** button to add a new row in which to define a new data field. Enter “Subsystem” for the **Field Name** and “7” as the **Size**, and leave the **Input Type** as “Text”.

The **Applicability** column determines what tables are assigned the data field (both existing tables and any new ones). The three types of applicability are “All tables”, “Roots only”, and “Children only”. “All

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 19 of 48

tables” means that any table of this type that’s created or exists is assigned the data field if it doesn’t already have it. For “Roots only” only root tables are assigned the field, but not child tables. Finally, “Children only” assigns the data field only when a child table is created. When the column is selected a combo box appears displaying the three choices. Select “Children only” from the list, then press the **Update** button to create the data field, which appears beneath the table in the table type editor (see Figure 22).

The image shows a software window titled 'Table type editor'. It has a tabbed interface with 'Lower Limit' and 'Lower limit' tabs. The 'Lower limit' tab is active, showing a 'Positive integer' data type. Below the tabs is a large text area labeled 'Description' containing the text 'Telemetry and data structure table definition'. At the bottom, there are two input fields: 'Telemetry message name & ID' (highlighted in yellow) and 'Subsystem'.

Figure 22. Table type with default field assigned

If a value is entered into the data field it is treated as a default value, so when a table of this type is created then the data field has the default value until changed by the user. Press the **Store** button to store the updates to the **Structure** table type. Note that the open table editor “myStruct” is unchanged; it’s a root table so the field doesn’t apply, and even if it were a child table it already has a field with this name so the new field wouldn’t be added. Close the table type editor.

3.7 Create a child table

A table is a child table if it is referenced by another table. In practice this is when a structure table uses another structure as a data type for a variable. Create a new structure table named “myChildStruct”. To do this use the **Data | New table(s)** command, then select **Structure** as the table’s type and enter the name “myChildStruct” in the **Create Table** dialog. Press the **Create** button to create the table. Edit the table by executing the **Data | Edit table(s)** command and selecting the table from the **Prototypes** or **Parents & Children** headings in the table tree. At this point the new structure isn’t a child since it isn’t referenced by another structure, so for now it meets the classification of a root table and the data fields displayed reflect this (i.e., the “Telemetry message name & ID” field, applicable for root tables is displayed, whereas the “Subsystem” field, applicable to child tables only, is not). Notice that the table is opened in a separate table editor than the one for the table “myStruct”. When opening a table or tables via the main window menu command a new table editor is created. If you want the table to open in an existing editor use that editor’s **File | Edit table(s)** command instead. Also, by hovering the mouse pointer over the table’s tab in the editor, then pressing and holding the left mouse button, the tab may be dragged to another editor. Releasing the mouse button causes the table’s tab to move to the selected editor.

Add a variable to “myChildStruct” as shown in Figure 23. Remember to press the **Store** button to save the change to the database.

The image shows a table editor window titled 'myChildStruct'. It contains a table with the following columns: Variable Name, Description, Units, Data Type, Array Size, Bit Length, Enumeration, Minimum, Maximum, Rate, and Lower Limit. The first row of data is: myVar3, (empty), (empty), Float, (empty), (empty), (empty), (empty), (empty), (empty), (empty), and a highlighted yellow cell in the Lower Limit column.

Figure 23. Variable added to structure “myChildStruct”

Select the table editor for “myStruct” and create a new variable “myChildVar”, then select its **Data Type** column. The combo box containing the data types appears and at the bottom of the list is the structure name “myChildStruct”. Select this structure as the data type, then store the change in the project database. The variable “myChildVar” is now a structure of type “myChildStruct” (see Figure 24).

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myArray1			Float	3						
myArray1[0]			Float	3						1
myArray1[1]			Float	3						2
myArray1[2]			Float	3						3
myVar1			uint16_t			0 Off, 1 On				4
myChildVar			myChildStruct							

Figure 24. Child structure variable added to structure “myStruct”

Use **File | Edit table(s)** in the table editor displaying “myStruct” to open the table tree, then select the “Expand all” check box. Under the **Parents & Children** heading the table “myStruct” now has a sub-structure, “myChildStruct.myChildVar”. The form of the child’s name is *data type.variable name*. Select the child table in the tree and press the **Open** button. The child is opened under a separate tab in the same editor as “myStruct”. Notice that the tab contains the name of the root table (“myStruct”) and the variable (“myChildStruct.myChildVar”). This is the format even when there are multiple nest levels of structures referencing structures; however, hovering the mouse pointer over the tab provides a tool tip showing the full path to the variable (as well as the tables’ type name). Compare “myChildStruct.myChildVar” to its prototype table, “myChildStruct”, in the other open table editor (see Figure 25).

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myVar3			Float							

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myVar3			Float							

Figure 25. A child table and its prototype

“myChildStruct.myChildVar” represents an instance of the structure “myChildStruct”. As such, it inherits the characteristics of “myChildStruct”, such as variable names, data types, array sizes, data fields (and their values), etc. Some of these characteristics are immutable; in other words, these can’t be changed in “myChildStruct.myChildVar” since doing so would make it no longer an instance of “myChildStruct”. The cells containing the non-changeable items have gray backgrounds. The remaining cells can be altered. An instance of a table always inherits the values in its prototype’s cells, but if altered in the instance then the instance’s values override the inherited values. Data fields (and their values) are inherited only at the point when the instance is created. Data fields added, removed, or altered in the prototype afterwards are not propagated to existing child tables.

To demonstrate inheritance of table values, in “myChildStruct” enter the text “myVar3 description” in the **Description** column for variable “myVar3”, then store the update. Notice that the description for “myVar3” in the editor for “myChildStruct.myChildVar” is automatically updated as well. Now edit the **Description** column in “myChildStruct.myChildVar” for “myVar3” and change it to “Custom myVar3 description”, and store the change. The contents of the two tables should appear as in Figure 26.

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myVar3	Custom myVar3 description		float							

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Minimum	Maximum	Rate	Lower Limit
myVar3			float							

Figure 26. Overriding inheritance of a table cell

Close the editor for “myChildStruct.myChildVar” by selecting the **Close** button for its editor (the editor window remains since the table “myStruct” is still open in it; closing “myStruct” would cause the editor window to disappear as well). Reopen “myChildStruct.myChildVar”, but this time use the short cut method: position the mouse pointer over the data type cell for variable “myChildVar” in “myStruct”, then double click the right mouse button; “myChildStruct.myChildVar” is opened in the same editor window with “myStruct” as before. Notice that “myChildStruct.myChildVar” retains the custom value in the **Description** column that was entered previously, and no longer inherits the description from the prototype. Edit the description in “myChildStruct.myChildVar” again, delete the text from the **Description** cell, then store it, then close and reopen myChildStruct.myChildVar”. The description remains empty and doesn’t inherit from the prototype since a blank cell is considered an overriding value as well.

To restore inheritance to a cell, select the cell, then in the editor’s menu bar select **Edit | Replace selected | with prototype**. The contents of the cell is replaced with that from the prototype (“myVar3 description” in this case), preceded by a special flag character as shown in Figure 27. Store the change to restore inheritance to the description cell (note that the special flag character is removed).

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
myVar3	myVar3 description		float					

Figure 27. Restoring a cell’s inheritance

3.8 Grouping tables

The number of data tables in a project can become large, making it more time consuming to locate a table and difficult to keep track of which tables are related. The group manager provides a means of more easily managing the tables. Open the group manager (Figure 28) using the main window’s menu bar **Data | Manage groups** command.

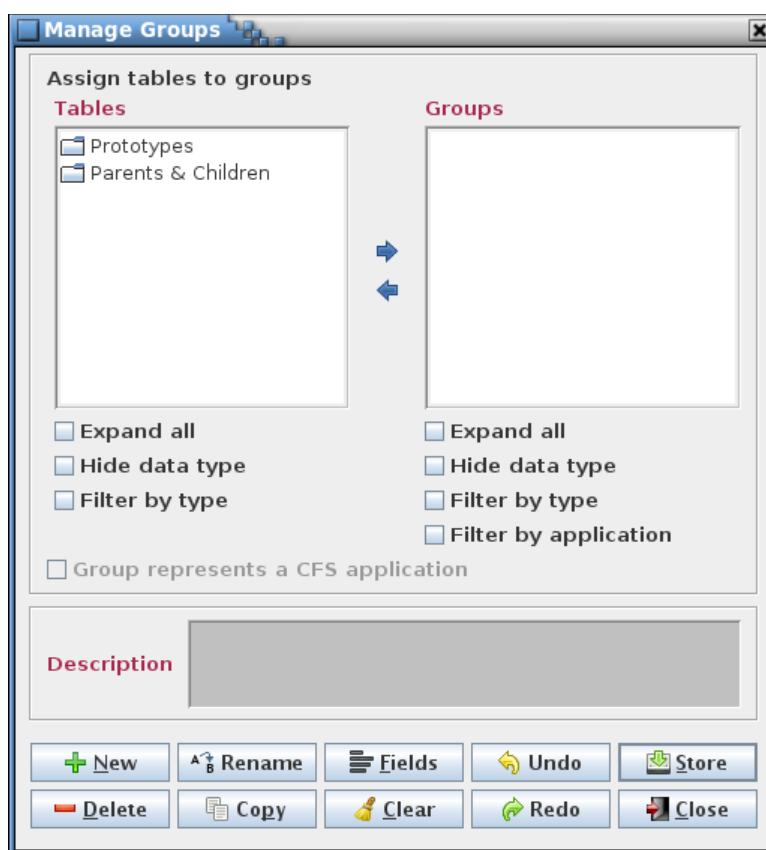


Figure 28. Group manager

The group manager allows tables, selected from the table tree in the upper left (and which acts just like the table tree when opening a table for editing), to be assigned to groups in the group tree in the upper right. A group must be created before a table can be assigned, so press the **New** button and in the dialog that appears enter the group name “My Group”, then press the **Okay** button. The group “My Group” appears in the group tree. Expand the table tree and from the **Parent & Children** branch of the tree select the table “myStruct” using the mouse or keyboard. In the group tree select “My Group”. Now press the right arrow button between the two trees – “myStruct” appears as a member of the group “My Group”. Notice that the child table of “myStruct”, “myChildStruct.myChildVar”, is included automatically. All tables that are descendants of a selected tree item are automatically included in the assignment. Figure 29 shows the result of assigning the table to the group.

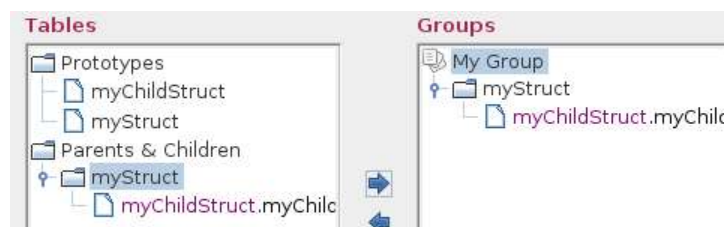


Figure 29. Assigning a table to a group

Any number of tables may be assigned to a group. Any number of groups may be created, and a table can belong to as many groups as desired. Deleting a group has no effect on the tables that were assigned to it.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 23 of 48

Create a second group, “Working tables”, and assign “myStruct” and “myChildStruct” to it from the **Prototypes** branch. The group manager appears as in Figure 30. Since prototype tables were specifically chosen there are no child tables added.



Figure 30. Multiple groups

Press the **Store** button to store the new groups in the project database, then exit the group manager by pressing the **Close** button. In the main window select **Data | Edit table(s)** to open the table tree for selecting a table to edit, then select the “Expand all” and “Filter by group” check boxes below the tree (see Figure 31; the dialog has been resized so that all of the tree is visible).

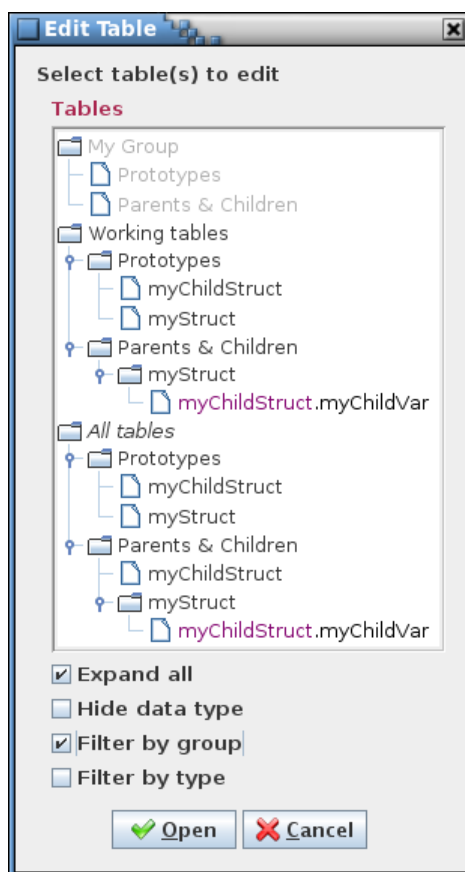


Figure 31. Table tree, filtered by groups

The tree’s original headings, **Prototypes** and **Parents & Children** are now subheadings for the group headings, **My Group** and **Working tables**. Under the group heading are the tables that are assigned to the group. Notice that a third group, **All tables**, also appears as a heading. This pseudo-group is automatically generated and displays every table. It becomes useful later, such as when associating

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 24 of 48

tables and groups with scripts (see paragraph 3.9). A table can be selected from the filtered tree just as when it isn't filtered – the filtering merely serves as an aid in arranging the tables in a meaningful way. Select **Cancel** to remove the dialog.

3.8.1 Group data fields

As with table types and data tables, data fields may be assigned to groups. These group data field assignments, as with table types and tables, can be accessed in the scripts and used in whatever way the user desires.

Select the group “My Group” from the group tree, then press the **Fields** button, which becomes active once a single group is selected. The data field editor appears; the editor behaves exactly as it does for the table type and table editors. Add the field as indicated in Figure 32.

Field Name	Description	Size	Input Type	Required
Group field		10	Text	<input type="checkbox"/>

Figure 32. Group data field

The group manager now appears as in Figure 33.

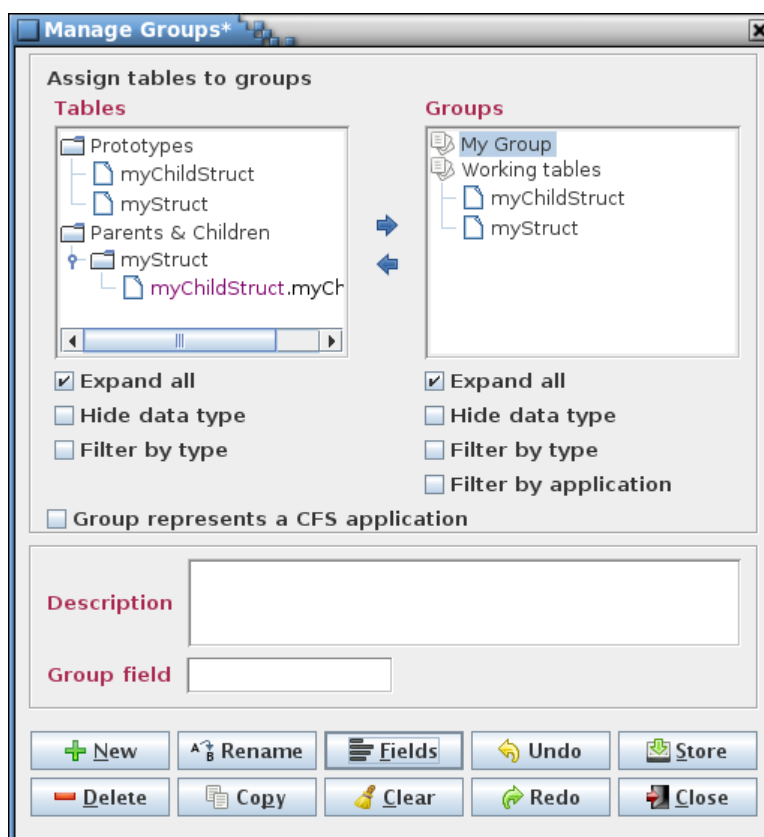


Figure 33. Group with data field assigned

Enter the text “My Group’s data field” into the “Group field” data field and store the changes. Close the group manager by pressing the **Close** button.

3.8.2 CFS applications

A specialized use of group data fields is in deeming a group as representing a CFS application. In the group manager, after selecting a group from the **Groups** tree, the check box “Group represents a CFS application” can be set. When this is done a number of data fields are automatically assigned to the group. The contents of these fields is used by the application scheduler and its companion script in order to create the CFS scheduler application’s (SCH) message definition and schedule definition tables. These fields can be modified, added to, or removed via the field editor as previously described.

3.9 Create a script association

In this section the data tables that have been created previously are “associated” with a JavaScript script. This association is then executed – the script is run using the data from the tables in order to produce an output file.

Associations are created using the script manager. In the main window command menu select **Script | Manage**; the script manager appears (Figure 34).

Figure 34. Script manager

The script manager is divided into five main parts: the script selection field and button, and the association name and description fields (upper left), a table tree (upper right), the table of associations (middle), and command buttons (bottom). At a minimum an association needs a script file. Not all

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 26 of 48

scripts need a data table explicitly associated with them, but usually an association needs to be assigned one or more tables. A special script for this tutorial is provided as part of the CCDD installation files. The script, tutorial.js, is found in the **scripts** folder, located in the folder in which CCDD is installed. Select the script by pressing the **Select...** button, then in the script selection dialog (Figure 35) navigate to the **scripts** folder and select the file tutorial.js.

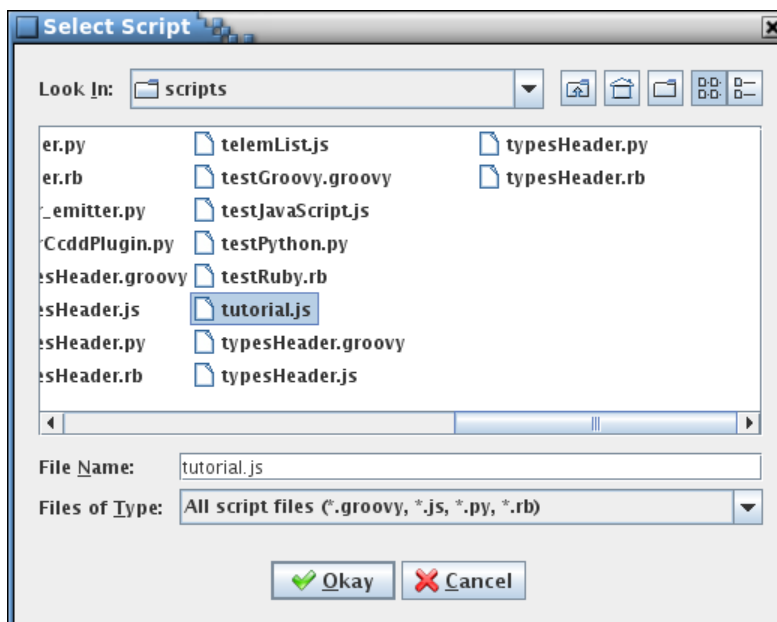


Figure 35. Script selection dialog

Press the **Okay** button. The script, including its full file path, appears in the script manager's script field. The table tree, located in the upper right of the script manager, acts the same as with other table trees already encountered in this tutorial. Expand the tree by using the mouse, keyboard, or by selecting the **Expand all** check box. Select the table "myStruct" from the **Parents & Children** heading, then press the **Add** button. The script associations table displays the new association (Figure 36). When a table is selected, all of the data from its child tables are automatically included, so these do not need to be explicitly selected when creating the association (though it doesn't create any problems if they are selected). In this case the data from the child table "myChildStruct.myChildVar" will be provided to the script along with that from "myStruct".

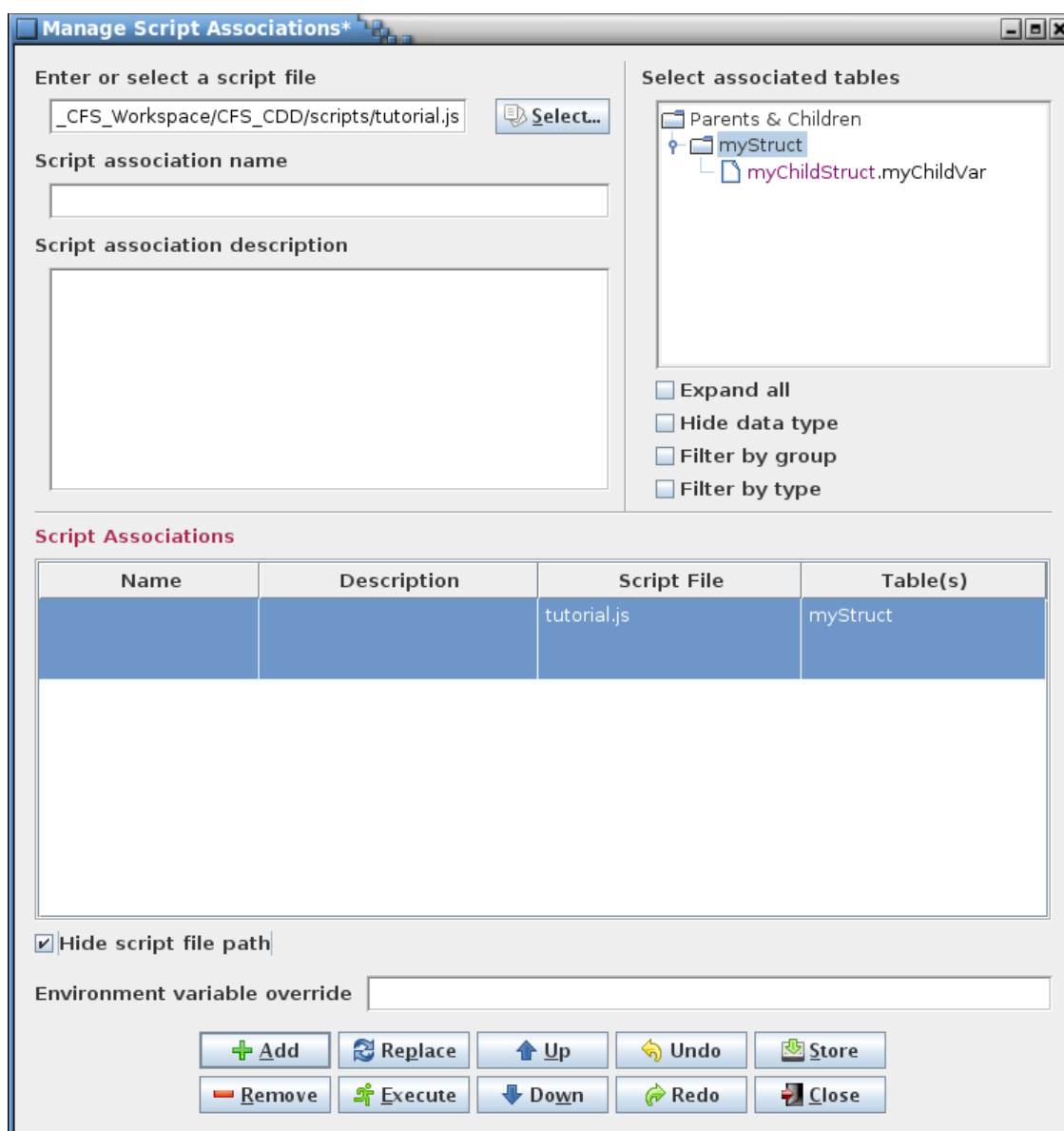


Figure 36. Script associated with a table

Select the **Filter by group** check box under the table tree and note that the two groups created earlier appear. A third group, “*All tables*”, also appears in the table tree; this group is created automatically and contains all tables. Notice that the group “*Working tables*” only contains “*myStruct*” and its child table, but does not show the table “*myChildStruct*”. This is because “*myChildStruct*” is only a prototype table, not a root or child table, and this table tree doesn’t display prototypes. Select the group “*My Group*” and press the **Add** button. A second association is added to the table (Figure 37). In the Table(s) column for the first association created it shows “*myStruct*”, but for the new association it shows “*Group:My Group*”. At this point these associations are functionally identical; if executed both access the data from “*myStruct*” and its child table “*myChildStruct.myChildVar*”. The advantage of assigning a group instead of a table is that if the group member tables are altered the association doesn’t need to be changed; all tables belonging to the group are accessed by the script. In the case where a table is

assigned to the script and another table is desired to be added, a new association must be created. The manager allows assigning both tables and groups to a script in a single association.

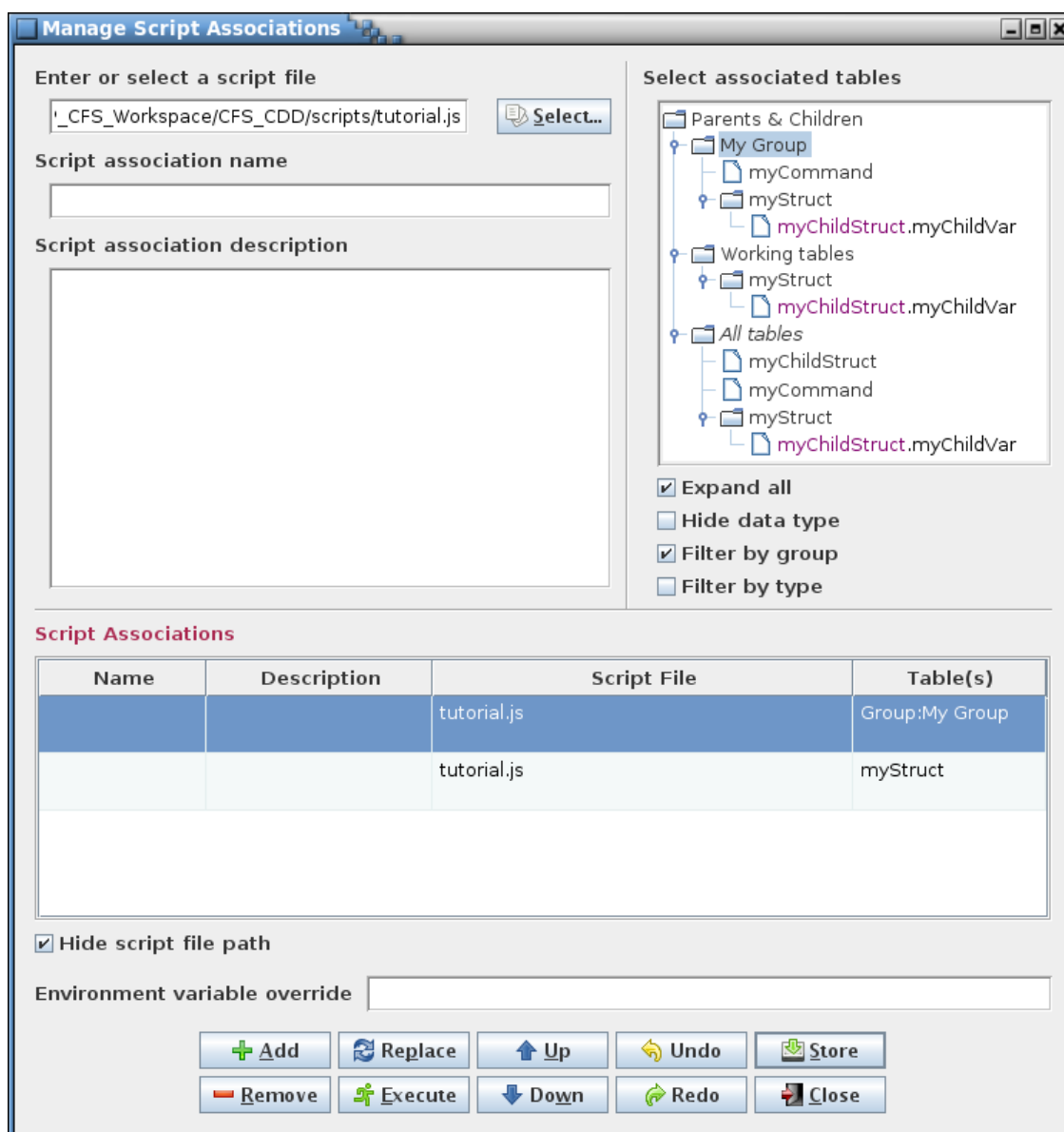


Figure 37. Script associated with a group

Press the **Store** button to store the associations in the project database. By storing the associations, these can be executed at a later time without having to recreate them.

3.10 Execute a script association

A script association may be executed from within the script manager or from the script executive (accessed using the main window **Script | Execute** command from the menu bar). The executive displays only the script association table and no changes can be made to the associations from this dialog. The manager and executive can't be open simultaneously; opening one closes the other.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 29 of 48

Select the association specifying the table “myStruct” using the mouse or keyboard. The entire line in the table is highlighted. Press the **Execute** button to execute this association. A dialog briefly appears while the script is running. This dialog allows halting execution of the script, returning control to the CCDD application. Halting a running script can be used, for example, to stop execution in the event the script contains an infinite loop.

The script, `tutorial.js`, creates an output file, `tutorial.output`, in the same folder from which the CCDD application was started. Use a file editor to locate and open the output file. Figure 38 shows the contents of the file.

```

/* Created : Wed Oct 25 09:43:59 CDT 2017
   User    : rmccune
   Project : MyProject
   Script  : /home/rmccune/COP_CFS_Workspace/CFS_CDD/scripts/tutorial.js
   Table(s): myStruct,
             myStruct.myChildStruct.myChildVar
   Group(s): My Group
*/

Table : myStruct
Type   : Structure
Description: My structure table

| Variable Name | Description | Units | Data Type | Array Size | Bit Length | Enumeration | Rate | Lower limit | | |
|---|---|---|---|---|---|---|---|---|---|---|
| myArray1 | | | float | 3 | | | | |
| myArray1[0] | | | float | 3 | | | | 1 |
| myArray1[1] | | | float | 3 | | | | 2 |
| myArray1[2] | | | float | 3 | | | | 3 |
| myVar1 | myVar1 description abc | | uint16_t | | | 0|Off,1|On | | 4 |
| myChildVar | | | myChildStruct | | | | | |

Data Fields:
Name: Subsystem Value: MySystem
Name: Message ID Name Value: MY_MSG_ID
Name: Message ID Value: 0x0900

Table : myStruct.myChildStruct.myChildVar
Type   : Structure
Description:

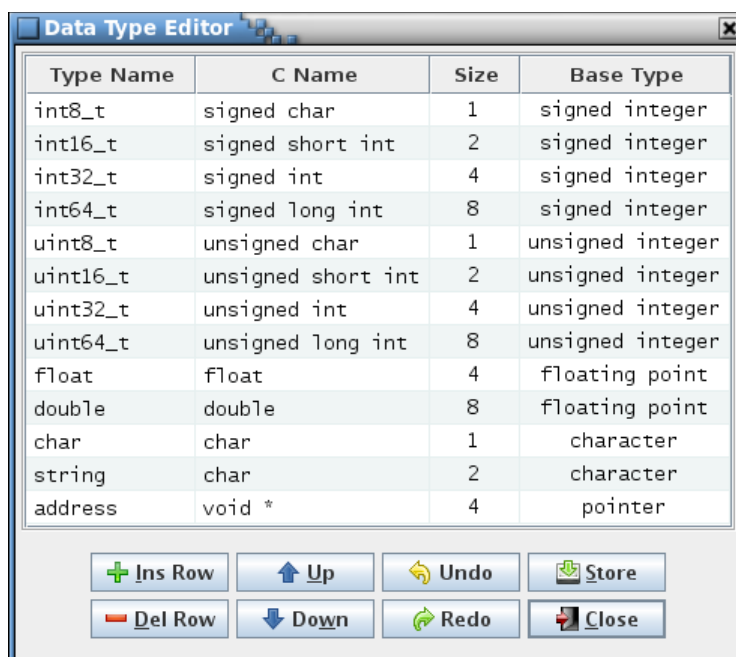
| Variable Name | Description | Units | Data Type | Array Size | Bit Length | Enumeration | Rate | Lower limit |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| myVar3 | myVar3 description | | float | | | | | |

```

Figure 38. Script output file

3.11 Data types

CCDD provides a means of changing the primitive data types. This can be as simple as changing the names of the existing primitives, or can include the addition of new types, such as structure pointers. The changes are made via the data type editor, accessed using the main menu command **Data | Manage data types**. When executed the dialog shown in Figure 39 is displayed. A number of primitive types are provided by default.



Type Name	C Name	Size	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	2	character
address	void *	4	pointer

Buttons: + Ins Row, Up, Undo, Store, Del Row, Down, Redo, Close

Figure 39. Data type editor

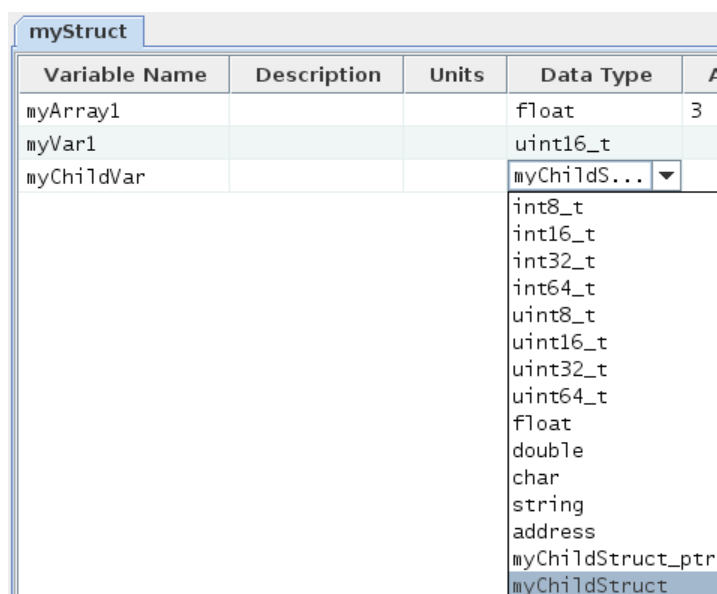
Each data type is constructed from one of five base types and a byte size. For example, the first primitive in the editor, `int8_t`, has a base type of “signed integer” with a size of 1 byte. The type and C names are user definable; a minimum of one of these names must be entered (the reason for having two names is so that the information can be used to create `#typedef` statements in a C header file created by a script). CCDD uses the **Type Name** (if present; if not, then the **C Name**) for display in the Data Type column combo boxes. If the existing names, sizes, or base types are altered, then when the changes are stored any existing use of the data type is updated in the data tables.

For the tutorial create a pointer to structure “`myChildStruct`”. Insert a new line into the editor table using the **Ins Row** button. Select “pointer” as the base type in the **Base Type** combo box that appears when the cell is selected, and “4” as the size. Edit the **C Name** column. The structure name, “`myChildStruct`”, can be typed directly, but instead press the Ctrl-S keys. This causes a combo box to appear listing all of the structure table names. Choose “`myChildStruct`” from the list. The asterisk, denoting a pointer, is automatically appended to the name if not entered by the user. Enter “`myChildStruct_ptr`” for the **Type Name**. The result should look like Figure 40.

Type Name	C Name	Size	Base Type
<code>myChildStruct_ptr</code>	<code>myChildStruct*</code>	4	pointer

Figure 40. New data type

Press the **Store** button, then close the data type editor. Open “`myStruct`” in a table editor if not already open. Select one of the **Data Type** column cells in order to show the combo box (Figure 41) and notice that the new pointer type, “`myChildStruct_ptr`”, appears in the list. The list always displays the primitives before any structure references, and the order of the primitives is the same as in the data type editor.



Variable Name	Description	Units	Data Type	Address
myArray1			float	3
myVar1			uint16_t	
myChildVar			myChildS...	

- int8_t
- int16_t
- int32_t
- int64_t
- uint8_t
- uint16_t
- uint32_t
- uint64_t
- float
- double
- char
- string
- address
- myChildStruct_ptr
- myChildStruct

Figure 41. Added data type

3.12 Macros

Macros are text strings that are used to represent a numeric or text value. The CCDD utility allows macros to be inserted within the data table cells. The benefit of using a macro is that it can be used across numerous tables, and if the value needs to change it can be done in a single place rather than in each individual table. Macros are created and updated in the macro editor (Figure 42), accessible via the main menu **Data | Manage macros** command. Each macro consists of a macro name and a value, which may be blank.

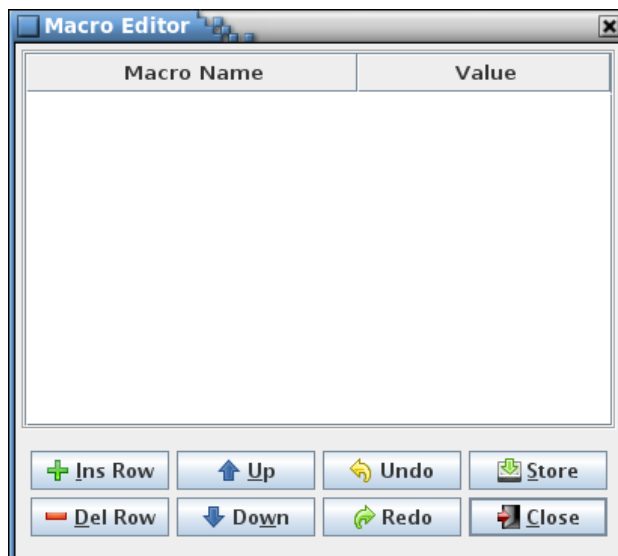


Figure 42. Macro editor

To add a macro, insert two rows into the macro editor table using the **Ins Row** button. In the first row enter “aNumber” in the **Macro Name** column and “5” in the **Value** column, and in the second row enter

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 32 of 48

“someText” in the **Macro Name** column and “abc” in the **Value** column (see Figure 43). Store the changes and close the macro editor.

Macro Name ^	Value
aNumber	5
someText	abc

Figure 43. Added macros

Open “myStruct” in a table editor if not already open. Edit the description column for the variable “myVar1”. Type “myVar1 description”, then press the Ctrl-M keys. A combo box appears displaying a list of macros (Figure 44).

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
myArray1			float	3				
myVar1	myVar1 description	aNumber	uint16_t			0 0FF,1 0n		4
myChildVar		aNumber	myChildStruct					
		someText						

Figure 44. Macro combo list

Select the macro “someText” from the list; the cell value displays “##someText##”, highlighted by a colored background. The leading and trailing “##” delimiting characters indicate to the application that a macro is in use. The macros can be typed directly (which must include the delimiter characters) in place of using the Ctrl-M sequence and combo box.

Expand the array “myArray1” by positioning the mouse pointer over an **Array Size** column cell and double clicking the right mouse button. Edit the array size for “myArray1” and delete the current value (3), then press the Ctrl-M keys. A combo box appears displaying a list of macros (Figure 45).

Variable Name	Description	Units	Data Type	Array Size
myArray1			float	aNumber
myArray1[0]			float	aNumber
myArray1[1]			float	3
myArray1[2]			float	3

Figure 45. Macro combo list; filtered

Notice that only the macro “aNumber” appears in the list. Macros with values that are invalid based on the column’s input type are filtered from the list; in this instance the macro “someText” has a value of “abc”, which is not valid in the **Array Size** column, which only accepts array index values (in the format #<#<,...>>). Select “aNumber” from the list and press Enter to end editing in the cell. Notice that the **Array Size** entries for “myArray1”’s array definition and members all show the macro, and that the number of members has expanded to 5, the value of the macro (Figure 46).

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
myArray1			float	##aNumber##				
myArray1[0]			float	##aNumber##				1
myArray1[1]			float	##aNumber##				2
myArray1[2]			float	##aNumber##				3
myArray1[3]			float	##aNumber##				
myArray1[4]			float	##aNumber##				
myVar1	myVar1 description	##someText##	uint16_t			0 0FF,1 0n		4
myChildVar			myChildStruct					

Figure 46. Array size change via macro update

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 33 of 48

Press and hold the keys Ctrl+Shift-M; the macros in the table are replaced by the macro values. Release the keys and the macro names are restored. Also, if the mouse pointer is hovered over a cell containing a macro a toll tip appears showing the cell value with the macro(s) replaced by the macro value(s).

Store the changes to the table, but leave the table editor open, and reopen the macro editor. Change the value for the macro “aNumber” from 5 to 3 and store the change; the array is restored to its former number of members.

3.13 Message IDs

CFS communicates within and between the core system and applications using messages. The message content depends on the information being transferred, such as telemetry values or commands. Each message is assigned an identification number, or message ID, that is used to identify the message to the listeners on the message bus so that the proper recipients can act on the message content. In general the IDs must be unique; otherwise a message could be mistaken for another and acted on erroneously. The CCDD application provides means for assigning message IDs (manually and automatically) and for detecting duplicate values. Input types are used to define a table cell or data field as containing a message ID or message ID name (recall these input types were used for the data fields created in section 3.5.6 of this tutorial), which CCDD can then use to locate these cells and fields.

IDs can be automatically assigned by selecting **Data | Message IDs | Assign IDs** from the main window menu. The dialog shown in Figure 47 appears.

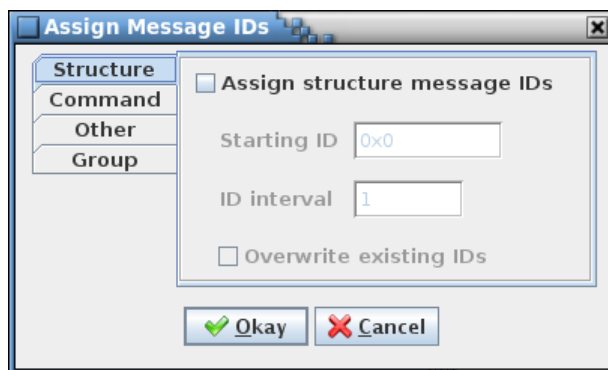


Figure 47. Assign message IDs dialog

The dialog has four tabs arranged down the left side. These allow assignment of IDs to the cells and data fields for a particular type of table (structure, command, or other table type), and to data fields associated with groups.

Using the “Structure” tab select the check box labeled **Assign structure message IDs**. This indicates that structure table message ID references are to be updated. The two fields, **Starting ID** and **ID interval**, are now enabled. IDs are assigned beginning with the starting ID; subsequent ID values are spaced according to the ID interval. Set **Starting ID** to “0x110” (the ID values are in hexadecimal format; the leading “0x” can be omitted). The **ID interval** is a decimal value and with it set as “1” the first ID assigned will be “0x0110” followed by “0x0111”, “0x0112”, etc. The Overwrite existing IDs check box determines if message ID cells and fields already containing a value should retain the present value or receive a new one. Since no ID has been assigned yet the setting for this box irrelevant. Select the **Okay** button to assign the message IDs to the structure tables.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 34 of 48

Open “myStruct” in a table editor if not already open. Below the table are the three data fields assigned earlier, one of which, names “Message ID”, has an input type of “Message ID”. Notice that the field, which had been empty, now contains the value “0x0110”.

Reopen the message ID assignment dialog and again check the **Assign structure message IDs** check box. For the starting ID enter “0x810”. Select the **Overwrite existing IDs** check box, then press the **Okay** button. Look again at the value of the “Message ID” data field for table “myStruct” – it shows “0x0900” and not “0x0810”. This is due to having ID values reserved.

In the main window execute the **Data | Message IDs | Reserve IDs** command. This produces an editor as shown in Figure 48.

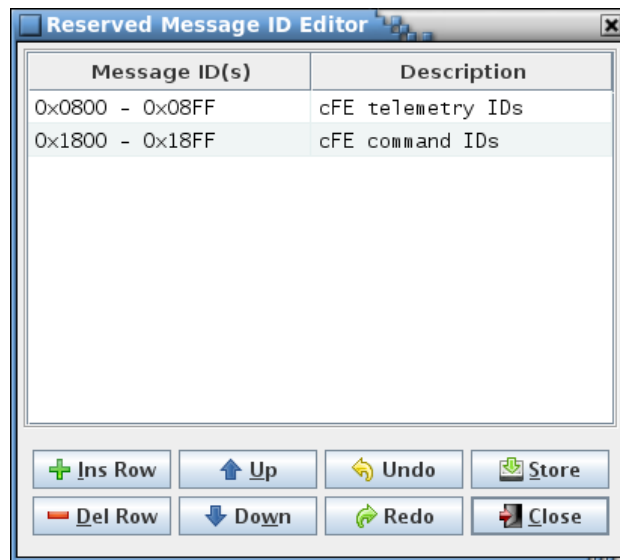


Figure 48. Reserve ID editor

The reserve ID editor allows defining single IDs or ranges of IDs that the automatic ID assignment feature ignores when assigning IDs. By default the ID ranges used by CFS telemetry and commands are reserved (these can be changed or deleted). In the example above the value “0x810” was entered as the starting ID value. The automatic assignment feature determined that this ID is reserved, so it added the interval value (1) to the ID to get “0x0811”. This value also falls within the reserved range, so the process is repeated until the value doesn’t fall within the reserved range, in this case “0x0900”, which is the first non-reserved value after the first reserved range.

3.14 Commands

3.14.1 Add a command argument to the Command table type

Command tables are used to store information pertinent to commanding the target vehicle or device. Command tables are created and edited exactly as is done for structure tables, except that the table type **Command** is selected in the **New Table** dialog (Figure 3) when a command table is created. Figure 49 shows the default command table definition in the table type editor.

The screenshot shows the 'Table Type Editor' window with the 'Command' tab selected. The table below represents the structure of a command table type.

Column Name	Description	Input Type	Unique	Required
Command Name	Command name	Command name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Command Code	Command code	Command code	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Description	Command description	Description	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Name	Command argument 1 name	Argument name	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Description	Command argument 1 description	Description	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Units	Command argument 1 units	Units	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Data Type	Command argument 1 data type	Primitive	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Array Size	Command argument 1 array size	Array index	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Enumeration	Command argument 1 enumeration	Enumeration	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Minimum	Command argument 1 minimum value	Minimum	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Maximum	Command argument 1 maximum value	Maximum	<input type="checkbox"/>	<input type="checkbox"/>

Below the table, there is a 'Description' field with the text 'Command table definition'.

At the bottom of the window, there are buttons for '+ Ins Row', 'Up', 'Undo', 'Store', 'Del Row', 'Down', 'Redo', and 'Close'.

Figure 49. Command table type

Two columns are required to distinguish a command table, one with an input type of “Command name”, and one of “Command code” (note that these two column definition rows are highlighted). A command generally has one or more arguments associated with it. The default type definition includes columns for a single command argument. Each argument requires at a minimum a column with the input type “Argument name”. The number of columns in the command’s table type definition with this input type determines the maximum number of arguments the command can handle.

Other columns can be added to the definition. These can apply in general to the entire command, or specifically to individual arguments. The position of the column definition determines how the column’s association is interpreted. Any column before the first argument name column is assumed to apply to the entire command. All columns following an argument name column (except the columns with input type “Command name” and “Command code due to their special input types) until the next argument name column or the end of the table type, are assumed to be associated with that argument name column. Looking at the default definition in Figure 49 the column **Description** applies generally since it precedes the first argument name column. All of the columns following the column **Arg 1 Name** apply to the first argument since these follow the argument name column definition and no other argument is defined.

Open the table type editor for the **Command** type. Insert a new row at the end of the table by selecting a cell in the last row and pressing the **Ins Row** button. Enter “Arg 2 Name” in the **Column Name** column. The column names have no bearing on deciding which argument the column belongs to, but it’s suggested that a consistent naming scheme be used. In the **Input Type** column select “Argument Name” as the input type. Similarly, add a data type column for the second command argument. Finally, add a column named “Arg 2 Mode” with the input type of “Positive integer”. Figure 50 shows the completed second argument rows. Store the changes.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: September 2018	Page 36 of 48

To save time when adding another set of command argument column definitions the table type editor's **Edit | Add command arguments** menu command can be used. This adds columns for the command argument name, description, units, data type, array size, bit length, enumeration, minimum, and maximum.

Command		Structure		
Column Name	Description	Input Type	Unique	Required
Description	Command description	Description	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Name	Command argument 1 name	Argument name	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Description	Command argument 1 description	Description	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Units	Command argument 1 units	Units	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Data Type	Command argument 1 data type	Primitive	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Array Size	Command argument 1 array size	Array index	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Enumeration	Command argument 1 enumeration	Enumeration	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Minimum	Command argument 1 minimum value	Minimum	<input type="checkbox"/>	<input type="checkbox"/>
Arg 1 Maximum	Command argument 1 maximum value	Maximum	<input type="checkbox"/>	<input type="checkbox"/>
Arg 2 Name		Argument name	<input type="checkbox"/>	<input type="checkbox"/>
Arg 2 Data Type		Primitive	<input type="checkbox"/>	<input type="checkbox"/>
Arg 2 Mode		Positive integer	<input type="checkbox"/>	<input type="checkbox"/>

Figure 50. Command table type with second command argument columns

3.14.2 Create a command table and add command information

Create a new table, "myCommand", based on the **Command** table type. See paragraph 3.3 on details for creating a new table. Open the table "myCommand" in the table editor (**Data | Edit table(s)**). Insert rows and enter the command information as shown in Figure 51.

Note that even though a command has columns for entering argument information, these may remain blank if the argument doesn't apply (such as with a "no-op" command). Also notice that certain columns are grayed out. The minimum and maximum value cells become active once a numeric data type (e.g., float, uint16) is entered. An enumeration only applies if the argument's data type is an integer (signed or unsigned).

Store the changes to the command table.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.2
	Date: <i>September 2018</i>	Page 37 of 48

myCommand													
Command Name	Command Code	Description	Arg 1 Name	Arg 1 Description	Arg 1 Units	Arg 1 Data Type	Arg 1 Array Size	Arg 1 Enumeration	Arg 1 Minimum	Arg 1 Maximum	Arg 2 Name	Arg 2 Data Type	Arg 2 Mode
NO_OP	0x00	No operation											
SET_PRESSURE	0x01	Initialize	int_pressure	Internal pressure	ata	float			0	50	ext_pressure	float	3

Figure 51. Command information added to table “myCommand”

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	<i>Version 1.0</i>
	Date: <i>November 2017</i>	Page 38 of 48

3.14.3 Execute a script using the command information

Open the group manager (**Data | Manage groups**) and assign the table “myCommand” to the group “My Group”. Store the change and close the group manager.

Open the script executive (**Script | Execute**). Select the script association that specifies the group “My Group” in the **Table(s)** column. Since a group is used instead of specific tables (as in the other association shown in the associations table) the association doesn’t have to be altered in order to access the information in the new command table. Execute the selected association by pressing the **Execute** button.

When the script completes execution open the output file `tutorial.output`, located in the same folder from which the CCDD application was started. Figure 52 shows the contents of the file. Notice that the contents of the new command table, “myCommand”, is included with the structure tables. Below the table definitions the information for each command and its arguments (if any) is displayed.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 39 of 48

```

/* Created : Mon Oct 30 10:03:10 CDT 2017
User      : rmccLune
Project   : MyProject
Script    : /home/rmccLune/COP_CFS_Workspace/CFS_CDD/scripts/tutorial.js
Table(s) : myCommand,
           myStruct,
           myStruct,myChildStruct.myChildVar
Group(s) : My Group
*/

```

Table : myCommand
Type : Command
Description:

Command Name	Command Code	Description	Arg 1 Name	Arg 1 Description	Arg 1 Units	Arg 1 Data Type	Arg 1 Array Size	Arg 1 Enumeration	Arg 1 Minimum	Arg 1 Maximum	Arg 2 Name	Arg 2 Data Type	Arg 2 Mode
NO_OP	0x00	No operation											
SET_PRESSURE	0x01	Initialize	int_pressure	Internal pressure	atm	float			0	50	ext_pressure	float	3

Table : myStruct
Type : Structure
Description: My structure table

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
myArray1			float	3				
myArray1[0]			float	3				1
myArray1[1]			float	3				2
myArray1[2]			float	3				3
myVar1	myVar1 description abc		uint16_t			0 0ff,1 0n		4
myChildVar			myChildStruct					

Data Fields:
Name: Subsystem Value: MySystem
Name: Message ID Name Value: MY_MSG_ID
Name: Message ID Value: 0x0900

Table : myStruct,myChildStruct.myChildVar
Type : Structure
Description:

Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
myVar3	myVar3 description		float					

Command: NO_OP
Code : 0x00
Arg 1:
Arg 2:

Command: SET_PRESSURE
Code : 0x01
Arg 1:
Arg 1 Name Value: int_pressure
Arg 1 Data Type Value: float
Arg 1 Description Value: Internal pressure
Arg 1 Units Value: atm
Arg 1 Minimum Value: 0
Arg 1 Maximum Value: 50
Arg 2:
Arg 2 Name Value: ext_pressure
Arg 2 Data Type Value: float
Arg 2 Mode Value: 3

Figure 52. Script output file with commands

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 40 of 48

3.15 Importing and exporting tables

In this section a table definition is created external to the CCDD application. The table is imported into a project, then exported for comparison purposes.

3.15.1 Create an import file

It can be convenient at times (for example, when access to the CCDD application is unavailable) to be able to define a table and then import the information into the application. CCDD allows importing table definitions from files. The supported file formats are comma-separated values (CSV), electronic data sheet (EDS), JavaScript Object Notation (JSON), and Extensible Markup Language (XML) Telemetric and Command Exchange (XTCE). For this tutorial a table will be created in CSV format using a spreadsheet application and then imported into the CCDD project as a new table. Note that CCDD also can import data into an existing table in the project.

CCDD expects the information provided in CSV format to be parsed into sections, which define what data follows. Examples of the sections are those for the table definitions, table type definitions, and data type definitions. Each section consists of the section tag, which appears alone on a row, followed by the section contents on subsequent rows. The CCDD user's guide provides details on the sections' contents. Empty rows or rows beginning with a # character (a comment line) are ignored.

Use a spreadsheet application and create a spreadsheet with the contents as shown in Figure 53.

	A	B	C	D	E	F	G	H	I
1	<u>_name_type_</u>								
2	newRoot,newStruct.childOfNew	Structure							
3									
4	<u>_description_</u>								
5	Imported table definition								
6									
7	<u>_column_data_</u>								
8	Variable Name	Description	Units	Data Type	Array Size	Bit Length	Enumeration	Rate	Lower limit
9	newVarA			int16_t			2 0 Open,1 Close		
10	newVarB			int8_t	4				
11									
12	<u>_data_fields_</u>								
13	# Field Name	Description	Size	Input type	Required	Applicability	Value		
14	CPU #		5	Positive integer	TRUE	All tables		1	

Figure 53. Table definition in a spreadsheet

Four sections are used to define the table in this example: _name_type_ (cell A1), _description_ (cell A4), _column_data_ (cell A7), and _data_fields_ (cell A12). The first section, _name_type_, contains the table name (including a path if this is a child structure) and the table type, and must precede the remaining sections. If a second table definition is included in the spreadsheet then another _name_type_ section indicates the beginning of its sections. For the tutorial the table's type is a structure (as indicated in column B2 in the figure). The table's name is `childOfNew`, the last name in the table path (column A2). `childOfNew` has `newStruct` as its prototype structure table, and is a member of the root table `newRoot`. Notice that definitions for the tables `newStruct` and `newRoot` are not provided. These tables are automatically created if not already present in the project.

The second section, _description_ (cell A4), is followed by the description of the table (row 5 in the figure). This section is optional.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 41 of 48

The next section, `_column_data_` (cell A7), defines the columns names and the contents of each row and column in the table. The first non-empty, non-comment row following the section tag is the column names (row 8 in the figure). Each subsequent non-empty, non-comment row has the column values for that row of the table (rows 9 and 10 in the figure). The next section tag or the end of the file determines the number of rows in the table.

The last section, `_data_fields_` (cell A12), defines the data fields associated with the table. Each non-empty, non-comment row following the section tag creates a new data field. Row 13 is a comment line showing the names of the data field inputs for easy reference. Row 14 creates a data field, named "CPU #", that accepts only positive integer values, requires an input, with an initial value of "1". The applicability input, which only applies to data fields associated with a table type definition, may be empty.

Once the data is entered, save the spreadsheet in CSV format with the name `tutorial.csv`. The result should look similar to that in Figure 54. Notice that many rows end in one or more commas. These represent empty columns added by the spreadsheet application so that every row has the same number of columns. These extra columns are ignored by the CCDD application during the import operation.

```

_name_type_,,,,,,
"newRoot,newStruct.childofNew",Structure,,,,,
'_description_',,,,,,
Imported table definition,,,,,
'_column_data_',,,,,,
Variable Name,Description,Units,Data Type,Array Size,Bit Length,Enumeration,Rate,Lower limit
newVarA,,,int16_t,,2,"0|Open,1|Close",,
newVarB,,,int8_t,,4,,,
'_data_fields_',,,,,,
# Field Name,Description,Size,Input type,Required,Applicability,value,,
CPU #,,5,Positive integer,TRUE,All tables,1,,

```

Figure 54. Spreadsheet saved in CSV format

3.15.2 Import the table

In the CCDD main window select the **Data | Import** command; the **Import Table(s)** dialog appears (Figure 55). The dialog displays import files in the current folder; the folder path can be changed as desired. Files for the four supported import types are displayed – the filter can be changed to show files of a single type. One or more files can be selected and the types of the chosen files do not have to be the same. For this tutorial select the `tutorial.csv` file created by the spreadsheet application.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 42 of 48

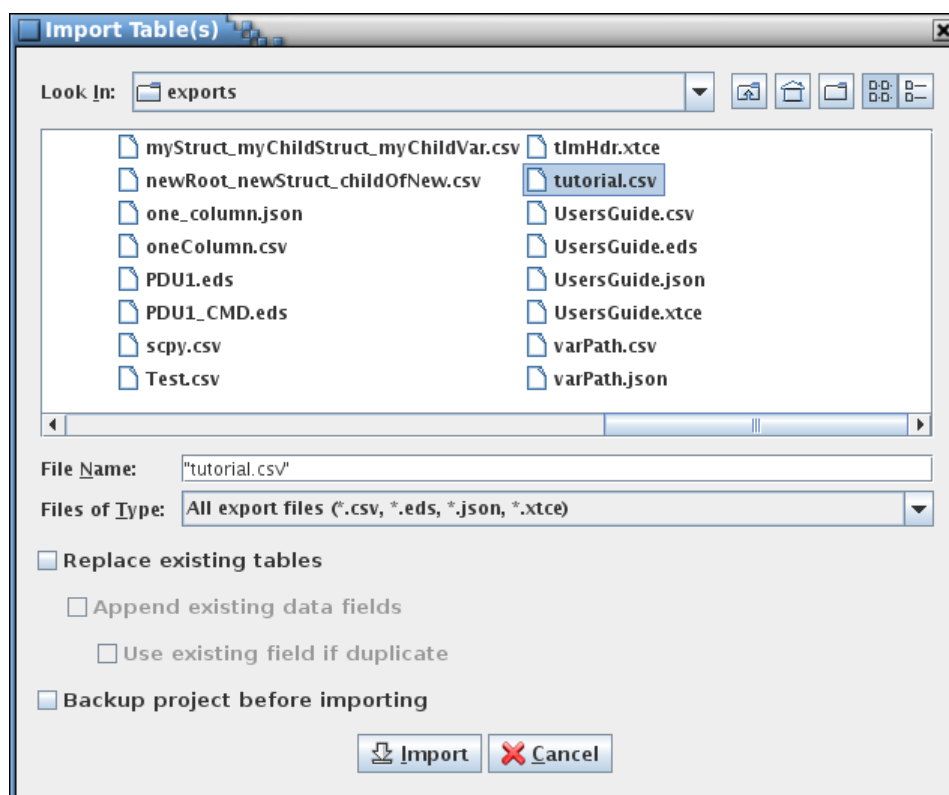


Figure 55. Import Table(s) dialog

Below the file selection controls are a number of check boxes. These control whether or not a table that already exists in the project is replaced by one of the same name in the import file (and how to treat data fields if replacement is selected), and if the project should be backed up prior to commencing with the import. Leave these unchecked for the tutorial. Press the **Import** button. A table editor appears as shown in Figure 56.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 43 of 48

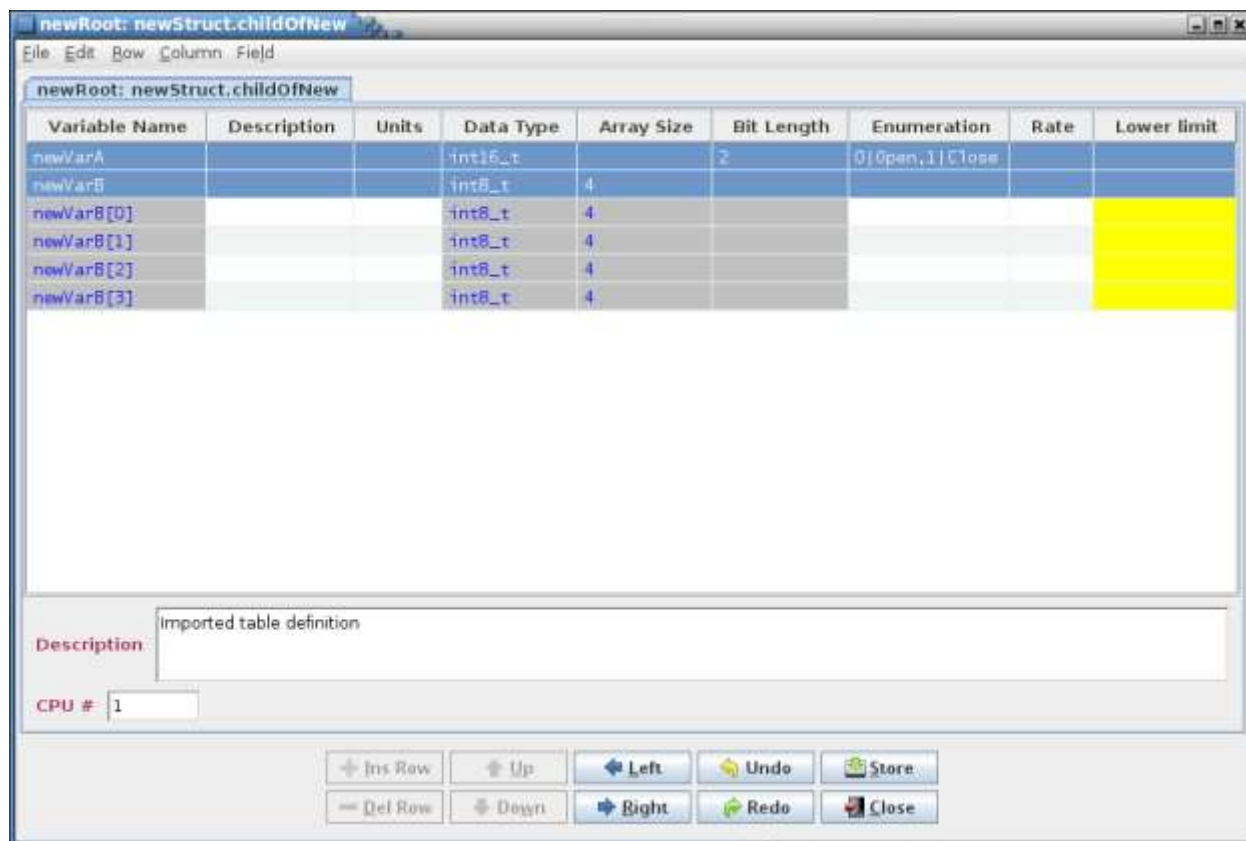


Figure 56. Imported tables in the table editor

Notice that the editor displays three tabs, one for each table created. The tab for the table **childOfNew** is initially selected. Since no prototype for this table existed in the project and wasn't supplied in the import file, the prototype, **newStruct**, was automatically generated. Select the tab for **newStruct** and compare it to **childOfNew**. The tables are identical in most respects, except that the **newStruct** has no description (since the description was assigned specifically to **childOfNew**) and only the prototype allows changing the variable names, data types, etc. The other table created, **newRoot**, is the parent table for **childOfNew**; it's also the root table for **childOfNew** (and a prototype table as well). Select its tab and note that it contains a single row that defines the variable **childOfNew** with the data type of structure **newStruct**.

Select the tab for **childOfNew**. In the CSV file the variable **newVarB** is defined as an array of type **int8_t** containing 4 members. The individual members are not specified in the CSV file; the members could have been entered into the import file, but since they weren't they are generated automatically during the import process.

Note that the table editor for **childOfNew** also displays the data field that is defined in the CSV file.

3.15.3 Export the table

Select the **File | Export table | CSV** command from the command menu in the open table editor for **childOfNew**. The Export Table(s) dialog, shown in Figure 57, appears.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 44 of 48

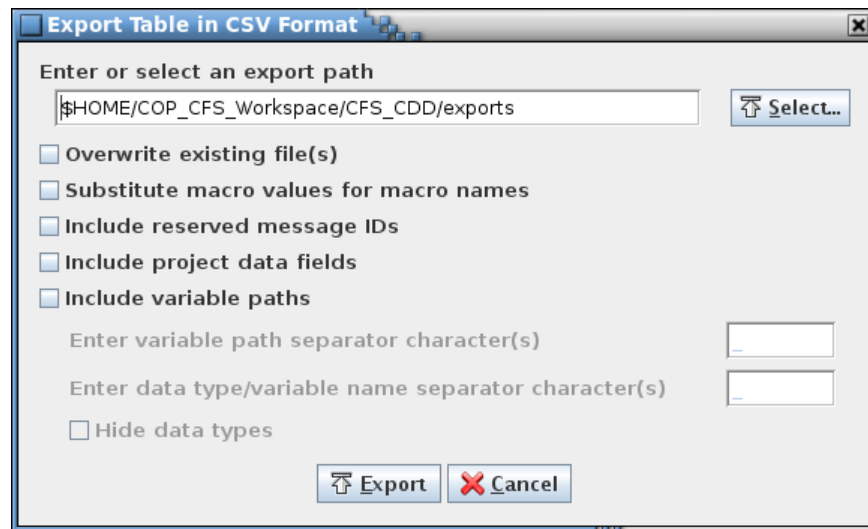


Figure 57. Export table(s) dialog

Enter the path for where the exported table's file should be placed, or press the **Select...** button to choose the path. The name of the export file is the table's path, but with the commas and periods replaced by underscores. In this case the file name is newRoot_newStruct_childOfNew.csv. The remaining check boxes can be ignored for this tutorial; see the user's guide for information on their use.

Press the **Export** button to export the table to the file. Using a text editor, open the export file. Its contents is shown in Figure 58.

```

1# Created Tue Nov 07 14:56:20 CST 2017 : project = MyProject : host = localhost:5432 : user = rmccune
2
3 name_type_
4 "newRoot,newStruct.childOfNew","Structure",""
5 description_
6 "Imported table definition"
7 column_data_
8 "Variable Name","Description","Units","Data Type","Array Size","Bit Length","Enumeration","Rate","Lower limit"
9 "newVarA","","","int16_t","","2","0|Open,1|Close","",""
10 "newVarB","","","int8_t","4","","","",""
11 "newVarB[0]","","","int8_t","4","","","",""
12 "newVarB[1]","","","int8_t","4","","","",""
13 "newVarB[2]","","","int8_t","4","","","",""
14 "newVarB[3]","","","int8_t","4","","","",""
15 data_fields_
16 "CPU #","","5","Positive integer","true","All tables","1"
17
18 table_type_
19 "Structure","Telemetry and data structure table definition"
20 "Variable Name","Parameter name","Variable name","true","true","true","true"
21 "Description","Parameter description","Description","false","false","true","true"
22 "Units","Parameter units","Units","false","false","true","true"
23 "Data Type","Parameter data type","Primitive & Structure","false","true","true","true"
24 "Array Size","Parameter array size","Array index","false","false","true","true"
25 "Bit Length","Parameter number of bits (bit values only)","Bit length","false","false","false","false"
26 "Enumeration","Enumerated parameters","Enumeration","false","false","false","false"
27 "Rate","Downlink data rate, samples/second","Rate","false","false","false","true"
28 "Lower limit","Lower limit","Positive integer","false","true","false","false"
29 table_type_data_fields_
30 "Subsystem","","7","Text","false","Children only",""
31
32 data_type_
33 "int8_t","signed char","1","signed integer"
34 "int16_t","signed short int","2","signed integer"

```

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 45 of 48

Figure 58. Table exported in CSV format

The table's definition appears first in the file. The information is almost identical to that in the import file that created the table. The differences are that the values are bounded by double quotes and the individual array members for the variable **newVarB** are included. The double quotes are included to account for any commas or other special characters within the individual values that could potentially affect parsing the file during an import operation.

Two other sections are included in the file. These define the table's type and data types of its variables. The export file may be used to import the table into another project, in which case it's important that the definitions are consistent between the projects. If the target project already has definitions for the table type and/or data types then these must match the ones in the import file in order for the import to proceed. If the table type and/or data types don't exist in the target project then these are created in the project and the import continues. Another section, containing macros definitions, is included if the table has any macro references and is handled in the same way as the table type and data type definitions.

3.16 Telemetry scheduling

Variables in the data structures can be assigned a rate (and possibly multiple rates if the structure has more than one column with the 'Rate' input type). This value is the variable's desired transmission or downlink rate in samples per second (for example, between a spacecraft and the ground station).

Telemetry scheduling is the assignment of telemetered variables to telemetry messages. The CCDD telemetry scheduler is the means by which this is accomplished. This telemetry scheduler provides performing the assignment automatically and manually, and ensures that each telemetry message isn't assigned more bytes than it can hold.

Once assignment is complete a script (provided as part of the CCDD package) uses the telemetry scheduler data to create the CFS housekeeping (HK) application's copy table.

3.16.1 Telemetry rates

The telemetry rates available for assignment to a variable are dependent on a number of parameters supplied by the user. If multiple telemetry streams are present then the parameters must be assigned for each stream. The number of available streams is equal to the number of unique structure table columns with a 'Rate' input type. By default the structure table type definition includes a single rate column. To adjust the rate parameters in the CCDD main window execute the **Scheduling | Rate parameters** command; the **Rate Parameters** dialog (Figure 59) appears.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 46 of 48

Figure 59. Rate Parameters dialog

Four parameters are required which in turn determine the available telemetry rates. The first two, **Maximum seconds per message** and **Maximum messages per second**, are common to each stream. Below these inputs are tabs, one for each stream. The tab names are the rate column names defined in the structure table type(s). A **Data stream name** can be assigned to the stream. This defaults to the column name, but can be changed as desired. The data stream name is used in the script access methods that require the stream reference. Two other parameters, **Maximum messages per cycle** and **Maximum bytes per second**, must also be entered. These may differ for each stream. The available rate, based on the input parameters, are display in the **Available rates** field.

Maximum seconds per message defines the slowest rates allowed; those less than once per second. Enter 5 as the value and press the tab key. The available rates displayed are once per second, once every 2 seconds, etc. up to once every 5 seconds – the value entered.

Maximum messages per second is the 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. Enter 4 for both **Maximum messages per second** and **Maximum messages per cycle** - the rates 4 and 2 are added to the **Available rates** field. Only those rates that are evenly distributed are displayed by default. Given the values entered so far it's possible to output a message three times per second (for example, in the first, second, and third messages of the four that are sent in a second), but the messages wouldn't be evenly time-spaced. If these unevenly spaced rates are desired, then the **Include unevenly time-spaced rates** check box should be selected. Select the check box and notice that a rate of 3 is now added. Deselect the check box and the 3 samples/second rate is removed.

The **Maximum bytes per second** is the total number of bytes that can be transmitted in a single second. This value divided by the **Maximum messages per second** is the maximum number of bytes that each telemetry message can contain.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	Version 1.0
	Date: November 2017	Page 47 of 48

The dialog values at this point should be the same as shown in Figure 60. Press the **Okay** button to accept the values and store them in the project's database.

Figure 60. Rate Parameters dialog with values entered

3.16.2 Assigning rates to variables

3.16.3 Linking variables

In certain instances it's critical that a message contains specific telemetry. For example, inertial measurement unit values for acceleration in each of three axes may be used to calculate a directional vector on the ground. If these values are returned in different telemetry messages the time difference between the messages needed to obtain the three values could produce in an error in the calculated result. The telemetry scheduler allows the variables to be assigned manually, so the user can control which telemetry values are in which messages; however this is prone to error. If automatic assignment is used then even this control is lost.

Linking ensures that specified variables remain together, so that if one variable is assigned to a message then any linked to it is assigned as well. The link manager, accessed via the main window's **Scheduling | Manage links** command, provides the means of linking variables together. The link manager appears as in

3.16.4 Bit length and bit-packing

The **Bit Length** column allows assigning a specific number of bits to for the variable to occupy. When variables of the same data type are created adjacent to one another in the table and both are assigned bit lengths then the application assumes these variables are packed together as a bit field within the same byte or bytes (dependent on the data type) if the combined length of the bits doesn't exceed the number of bits for the data type.

Johnson Space Center Engineering Directorate	Core Flight System Command and Data Dictionary Utility Tutorial	
	Doc. No. -----	<i>Version 1.0</i>
	Date: <i>November 2017</i>	Page 48 of 48

3.16.5 Creating the housekeeping copy table