QDTE User Manual

For version 1.2.8

29.Apr.2024

Contents

Introduction

QDTE is a tool that allows users to edit DeviceTree Binary blob (DTB) and DTB's inside of DTB ELF files. It supports adding and removing properties and nodes, modifying the values of properties, saving the resultant DeviceTree as a DTB file or DTS (DeviceTree source) file, and re-exporting XBLConfig files. It also provides some functionality for debugging in the form of a "raw view" of a DTB file that has been opened by the editor. It Also supports to read DTB elf from device and write back the new DTB elf to device.

Launching and Requirements

QDTE requires Python 2.7.6 or later. For Read/write DTB elf from device, QDTE requires QUTS installation on PC.

Opening DTB Files

To begin, use the File->Open DTB… menu or press Control+O to open a valid DTB file. If the file is invalid or corrupted, an error message will be shown to the user in a dialog box. Once the file has been loaded successfully, a Tree View should appear and be populated with the elements in the DTB. If the file has changed on disk while the DTGUI tool is open, it is possible to use the File->Reload File menu option to discard all changes and re-read the file from disk.

The Tree View

The Tree View displays the DTB visually in the form of a tree. There are four columns in this view which provide additional information on each item in the tree. The first column contains the full path to the item. The second column contains the type of the item in the device tree. This can be either a Node (which may have more nodes or properties within it) or a Property (which may have a value). The third column describes the data type contained inside of a property; for nodes, this column is blank. The fourth column contains the value of the property represented according to its data type; for nodes, this column is also blank.

In order to explore the children of a node, press the plus sign to the left of the node's path, or double-click the node. Double-clicking a property will open a dialog that allows the user to edit its value. This process is described in the Editing Properties section.

Node and Property Names

Names of nodes and properties must contain only characters that are a subset of `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,._+?#-`. Nodes may additionally have a name containing @ signs. Additionally, a node may not have two children with the same name (regardless of if the children are both nodes, both properties, or one and the other).

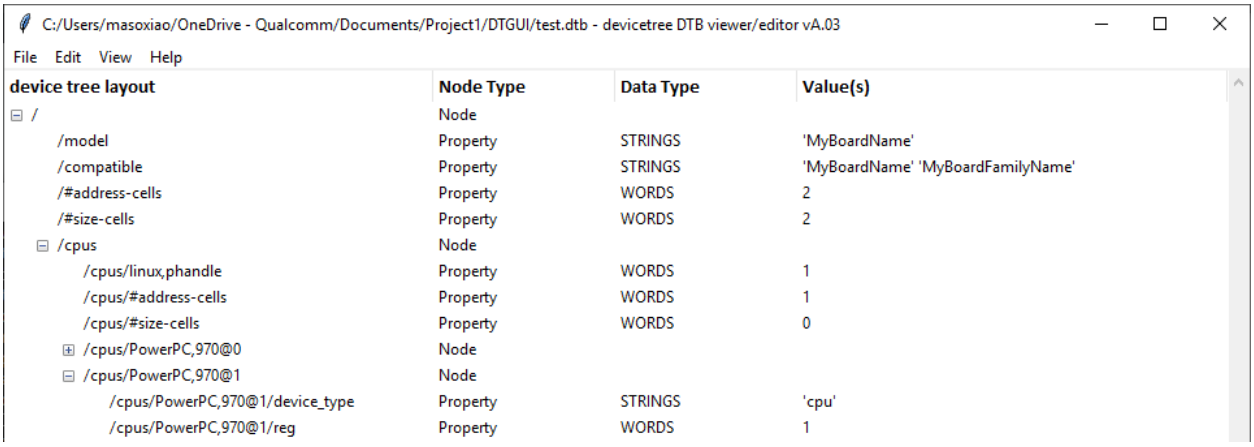


Figure 1: An example file opened in the TreeView, demonstrating how the four columns are populated (note that Nodes have no value in the Data Type and Value, but Properties do) and the plus/minus signs to the left of nodes with children (for example, `/cpus/PowerPC,970@0` is collapsed, so it has the plus sign to expand it, but `/cpus/PowerPC,970@1` is expanded, so it has the minus sign to collapse it).

Property Data Types

There are four possible data types, STRINGS, WORDS, NONE, and BYTES. DTB files do not explicitly state what type of value each property contains, so when loading a file, these types are automatically detected. Following are the criteria for each data type.

STRINGS

If the value of a property can be encoded as ASCII characters, is terminated with a NULL (0 byte), does not have two adjacent NULL bytes (i.e., does not contain any empty strings), does not contain newlines or carriage returns, and only contains printable characters (defined as any subset of `0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\x0b\x0c`), then the value will be treated as a STRINGS. This type corresponds to the `<string>` or `<stringlist>` types in the DeviceTree specification. Values of this type will display as a space-separated list of strings with single quotes surrounding them.

WORDS

If the property value is not a STRINGS, but the length of the property value is a non-zero multiple of four, then it is considered a WORDS type. Note that, although the DeviceTree specification allows for 32- and 64-bit integers, there is no actual way to distinguish between these two types. As such, all WORDS are 32 bits wide (`<u32>`); `<u64>` is simply two adjacent `<u32>` values instead (in big endian). WORDS may also correspond with the `<phandle>` type from the DeviceTree specification. Values of this type will display as a space-separated list of unsigned integers in either base 10 or base 16 (in which case the values will be prefixed with 0x prefix in order to clarify that they are shown in hexadecimal).

BYTES

The catch-almost-all type is BYTES. If the property has a non-empty value but does not fit the criteria for STRINGS or WORDS types, then it is considered a BYTES type. This type may correspond with the

<prop-encoded-array> type in the DeviceTree specification. Values of this type will display as a space-separated list of unsigned integers in either base 10 or base 16 (in which case the values will be prefixed with 0x prefix in order to clarify that they are shown in hexadecimal).

EMPTY

If the property has no value, then it is considered of the type EMPTY. This corresponds with the <empty> type in the DeviceTree specification.

/randomnode	Node		
/randomnode/string	Property	STRINGS	'stuffstuff'
/randomnode/blob	Property	BYTES	10 11 12 13 222 234 173 190 239
/randomnode/ref	Property	WORDS	2
/randomnode/empty	Property	EMPTY	

Figure 2: Example of each of the property types and how they are displayed in the DeviceTree.

Changing Value View Format

By default, numbers (WORDS and BYTES) will display in decimal in the Tree View. To view the values in hexadecimal, use the View -> Values As… menu option and click on the desired display format.

Editing Properties

Right clicking a property and selecting the "Edit…" option or double-clicking a property will cause a dialog window to appear that will allow the user to edit the name and value of the property. While editing a property value, no other elements of the GUI will be responsive other than the dialog box.

For WORDS and BYTES types, this dialog box will include the ability to view and edit the values in both decimal and hexadecimal formats. Editing the value in one text entry box will automatically synchronize the updated value to the other box. WORDS and BYTES are treated as unsigned integers, so a single WORDS value will have a maximum value of 4294967295, and a single BYTES value will have a maximum value of 255.

A single STRINGS value is any of the permissible characters in a string enclosed between two single quotes. In order to have a single-quote in a value, escape it with a backslash (thus, ' changes to \'). In order to have a backslash in a value, escape it with a backslash, too (thus, \ changes to \\). Unrecognized other backslash escape sequences will be left untouched in the string.

If multiple values are desired, delimit the values with a space. This is applicable for WORDS, BYTES, and STRINGS types.

To edit the name of a property, modify the portion of the property path text field after the last forward slash. Besides the property's name, no other portion of the path can be modified. The new name for a property is subject to the restrictions on names of nodes and properties mentioned previously. If both the name and value of a property are modified, it will be split into two operations;

first, the value of the property will be updated, and then the property will be renamed. Thus, two undo steps are necessary to fully erase the effects of an edit value and rename.

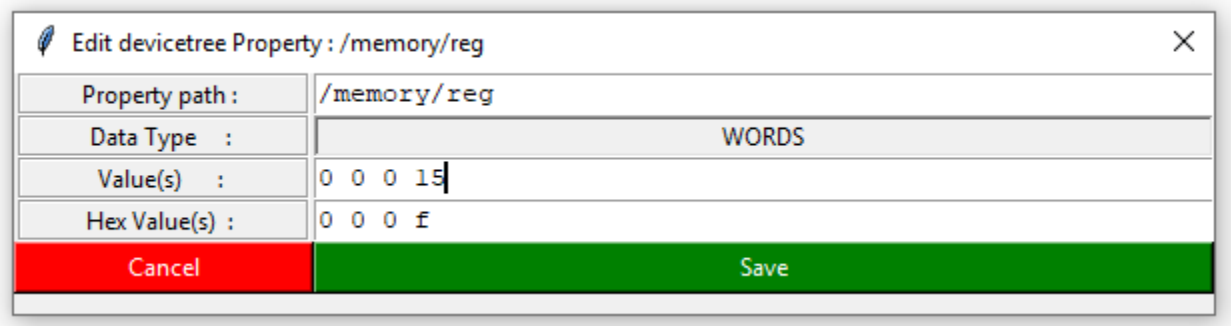


Figure 3: An example edit dialog for a WORDS type. Note how multiple values are delimited by spaces, and that both the decimal value and hexadecimal value are visible and editable for WORDS types.

Inline Editing

For quick edits of just a property's value, it is also possible to modify values without popping out a new window. To use inline editing, press tab (two presses may be required sometimes) or click on a property to select it and then click the value area (note that clicking too quickly may count as a double-click and open the edit dialog window instead). In inline editing mode, STRINGS values can be modified as they would be in the edit dialog, but WORDS and BYTES must be edited in the current value display mode (decimal editing for decimal view, hexadecimal editing for hexadecimal view; note, however, that hexadecimal editing does not require the `0x` portion of the number). To commit changes, press any of Enter, Tab (which will also jump to editing the next visible property value), Shift+Tab (which will also jump to editing the previous visible property value), or click anywhere outside of the box. To cancel changes, press the Escape key.

Adding Tree Items

Right-clicking a node will show the option to "Add New Child…" a "Node" or "Property of Type". Clicking on the "Node" option will prompt the user for the name of the new node and add it. Invalid node names will result in an error.

Clicking on the "Property of Type…" option will expand another menu that allows the user to choose the type of property to add. Doing so will pop up an Edit dialog corresponding to the desired type of the new property, with the difference that, upon confirmation, a new property of the specified type will be added, and if the dialog is cancelled, then the property will not be added.

Copying Tree nodes

Right-clicking a node will show the option to "Copy Node". Clicking on the this option will prompt the user for the path of the new node in the device tree. If the path of the new copied node exists, the

copied node is notified that this path does not exist and whether the user would like to create nodes along this new path "The path you requested does not exist. Create this path?". If the user agrees, the new path of nodes are created along with the new copied node. Invalid node names will result in an error.

Deleting Tree Items

All items in the DeviceTree, apart from the root node, may be removed. This functionality may be accessed by right-clicking on any item and clicking the Delete option. Removing a node that has children will also remove the node's children.

Undoing and Redoing Changes

If there has been a mistaken change made to the DeviceTree, it is possible to undo the change by pressing Control-Z or using the Edit->Undo menu option. To redo a change, press Control-Shift-Z or use the Edit->Redo menu option. Modifications to highlighting are not considered changes that can be undone or redone.

Locating Items

Pressing Control+F or using the Edit->Find… menu option will display a window that allows the user to find items inside of the DeviceTree that contain a given query text. There are currently three options for finding items: firstly, whether to search for the text in the names of items (note that, if a node name matches the query, all of its children will also match); secondly, whether to search for the text in the value of each item (where the value is what is displayed in the Tree View); and thirdly, whether matches should be case-sensitive. Pressing the "Find Next" button in the dialog or the F3 key in the main window will search the DeviceTree for the query and display the result, if there is one, in the Tree View and jump to the corresponding row in the Raw View if it is open. Results will wrap around to the top of the tree upon reaching the bottom.

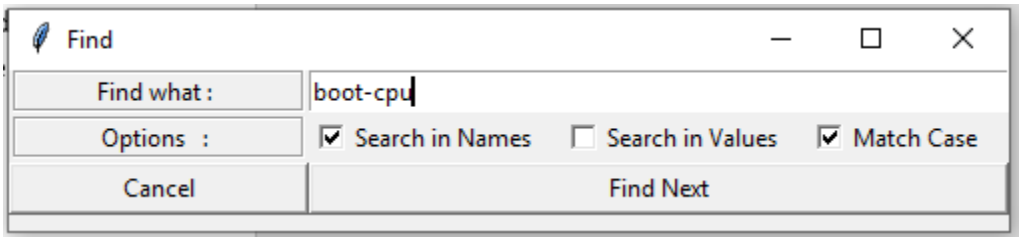


Figure 4: Example Find window where the user is searching for the string "boot-cpu" in the DeviceTree. Note how the "Search in Names" checkbox is checked, but the "Search in Values" box is not, so results will only be from the name of the item, and not the value.

Raw View

To display the view of the DTB file as a binary hex dump, use the View->Raw menu. This view displays exactly what an exported DTB file would look like.

There are three columns in the raw view. The first is the offset into the file, in hex. The second is the hex value of each byte. The third column is the character corresponding to each byte. Unprintable characters are replaced with a period (.) instead.

Highlighting

In order to facilitate debugging and reading through the raw view, it is possible to highlight specific items in the DeviceTree. To do so, right click on any item in the Tree View and choose the "Highlight…" option followed by any color (if the default options are not sufficient, the last option, "Custom…", will open a prompt for the user to enter whatever color they desire). Highlighting an item will highlight itself and all of its children (including ones that may be added later). The highlight will include the 4-byte start and end markers in addition to the data between the two markers, but for properties, it will not include the name of the property in the string table. Highlighting the root node will highlight every item in the structure block of the Device Tree, but exclude the header, memory reservation block, and strings block.

To clear all highlights, it is possible to use the View->Clear Highlights menu option.

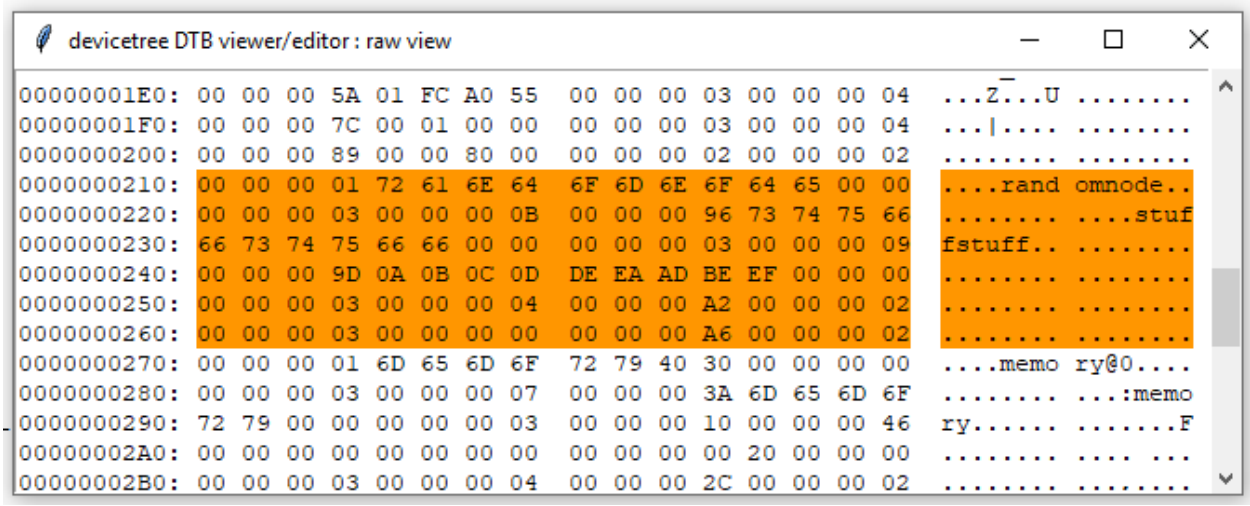


Figure 5: A screenshot of the Raw View showing the offset, hex value, and corresponding character columns. The node that was shown in Figure 2 is highlighted in orange here.

Saving DTBs

There are several potential options for saving the modified results.

The primary method to save modified results is to use the "File->Save" option (Control+S), which will overwrite the existing file. Alternatively, the user may use the "File->Save Copy As…" option (Control+Shift+S), which will prompt the user for a file to save the DeviceTree blob to. When saving DTB

files, items that cannot be modified in this GUI (e.g. the memory reservation block) will remain the same as the input file. In general, the output binary should be identical to what can be seen in the raw view.

It is also possible to save the modified results using the "File->Export to DTS" option. The user will be prompted for a file to save a DeviceTree source file to. When re-compiled with `dtc` or related tools, this source file should generate a similar structure to exporting the DeviceTree blob directly. The advantage of this method is that other parameters that cannot be modified in this tool can be changed. However, the disadvantage is that the file must be re-compiled.

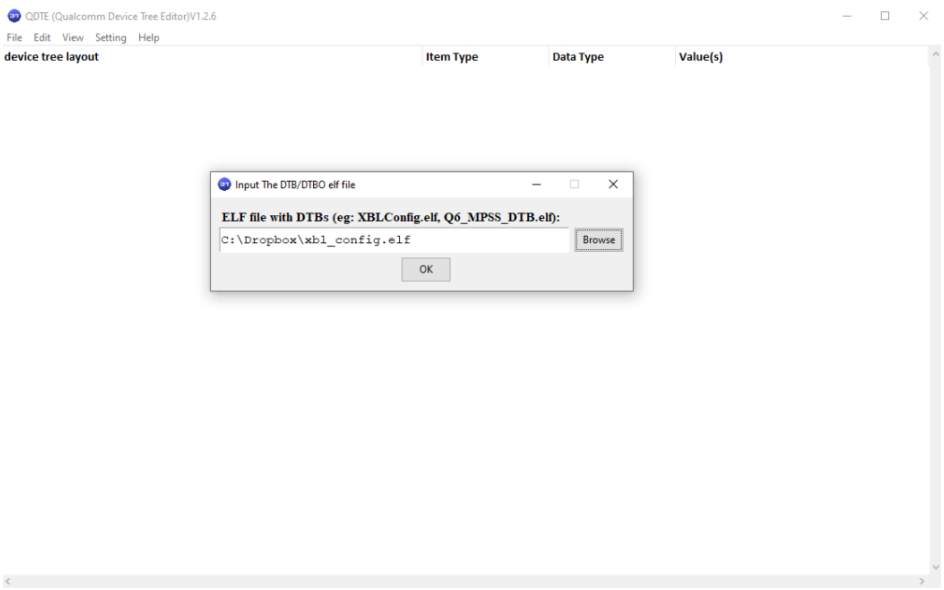
Finally, a human-readable JSON file summarizing the changes made may be generated using the "File->Export Change Report…" menu option. This file will contain a list of every change that occurred to the DeviceTree in the session, as well as the resultant hash of the DTB after the operation. More information on the export format can be found in the section titled Change Reports.

DTB ELF Files

In order to be able to disassemble and reassemble DTB ELFs, a path to the sectools must be specified for QDTE. To Open Setting menu to config it.

Opening and Editing DTB ELF Files from Build

To open an DTB ELF from build for editing, use the File->Open DTB Elf ->From Build … menu. Then it will prompt a window for input the path of DTB elf file.



After the selecting DTB elf file, the DTB ELF Helper window will appear containing, in order from top to bottom, helpful instructions for the user, the file name that is being edited, a list of DTB's found in the DTB ELF, a button to save the modified DTBs to DTB ELF file, and a console window.

Behind the scenes, the QDTE is calling into the Gen DTB elf toolchain to disassemble the ELF. As a result, it may take a few seconds before the console outputs a success message. Progress updates can be tracked in the console window. Once the DTB elf has been successfully disassembled, a list of DTB's found in the DTB ELF should appear, and a green "Success!" message should appear in the console. Note that the console window uses colors to convey the severity of messages; blue messages are informational messages, green messages indicate success, orange messages indicate warnings, and red messages indicate errors. Black messages contain miscellaneous debugging information output by the DTB ELF toolchain and are normally hidden.

To edit a file, double-click it, and it should appear in the primary QDTE interface. The DTB may mostly be edited as if it were any normal DTB file on the system. One major difference is that the user does not need to save the DTB file as they modify it; changes will automatically save when the user reassembles the DTB ELF. Additionally, because of the auto-saving, it is also not possible to reload the file from the filesystem.

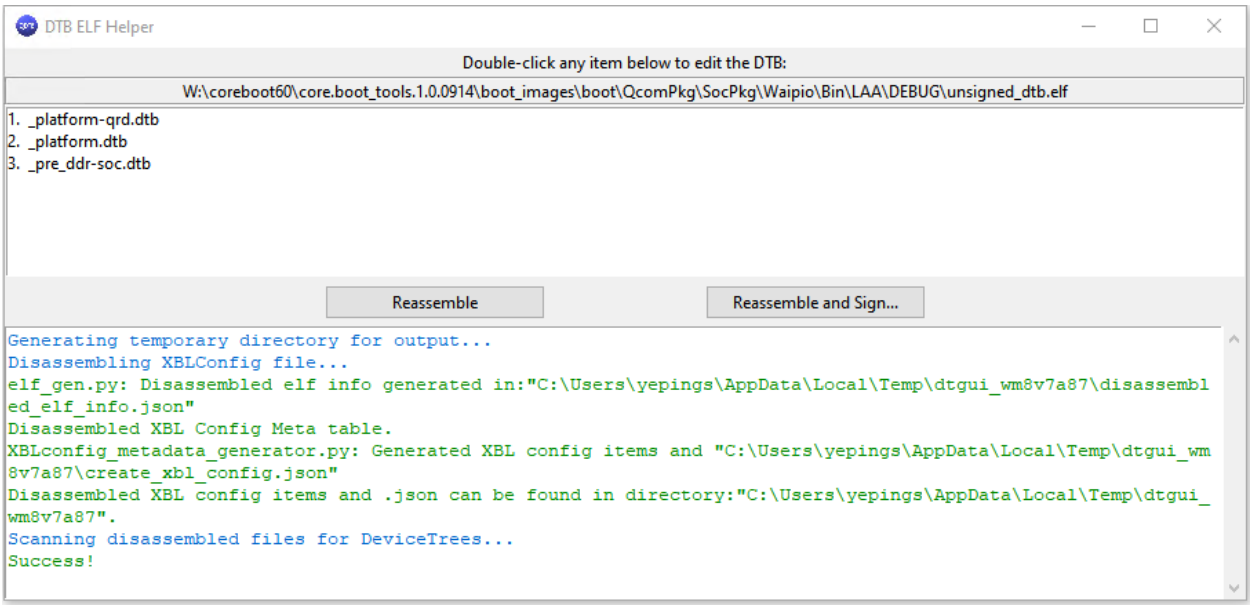
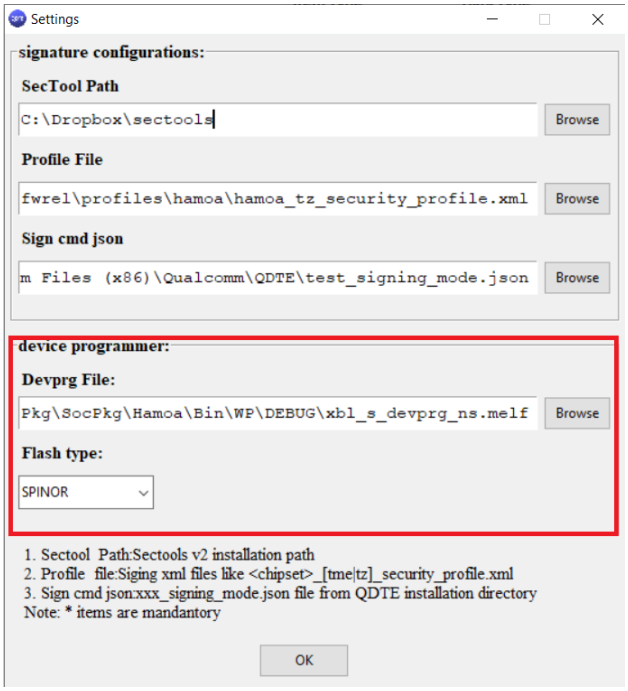


Figure 6: A sample DTB ELF Helper window showing the disassembled results of an XBLConfig file with Three DTBs inside of it.

Opening and Editing DTB ELF Files from Device

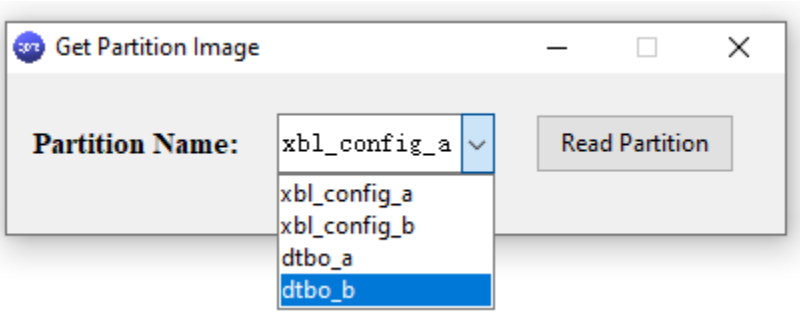
In order to open DTB ELF from Device, user must install the latest QUTS on the PC. QDTE will read/write DTB ELF file from device by QUTS. QUTS (Qualcomm Unified Tools Services) can be fetched from QPM.

Before read DTB elf from device, chipset specific deviceprovgrammer elf file and storage type should be configured correctly in settings menu.



To open an DTB ELF from device for editing, device must be in EDL mode. Choose File->Open DTB Elf ->From Device … menu.

QDTE will prompt Get Partition image GUI for select which DTB elf user want to read from device. For bootloader, user should choose xbl_config_a or xbl_config_b.



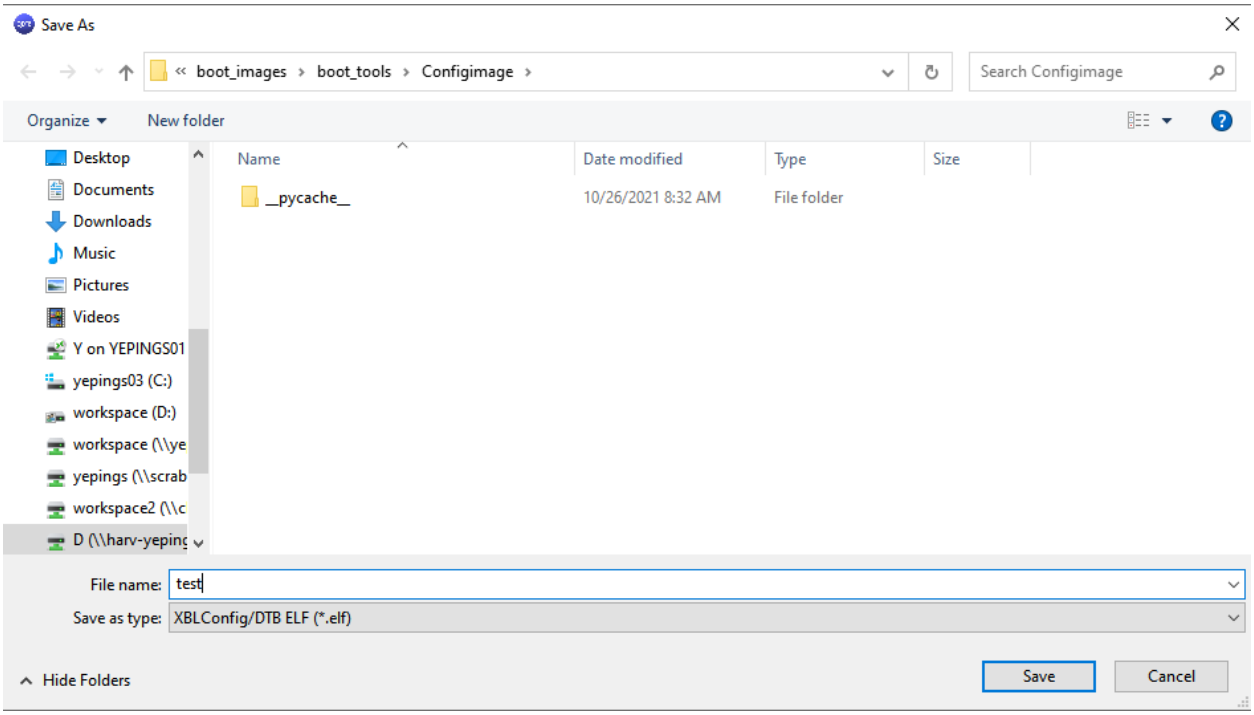
Once partition name was chosen, click “Read Partition” button. Once reading successfully, it will prompt “DTB ELF Helper” GUI.

Saving DTB Files

Once editing is complete, the user must save the DTB from within the primary QDTE window, return to the DTB ELF Helper window, and additionally save the modified DTB ELF file using the “Reassemble” or “Reassemble and Sign…” button.

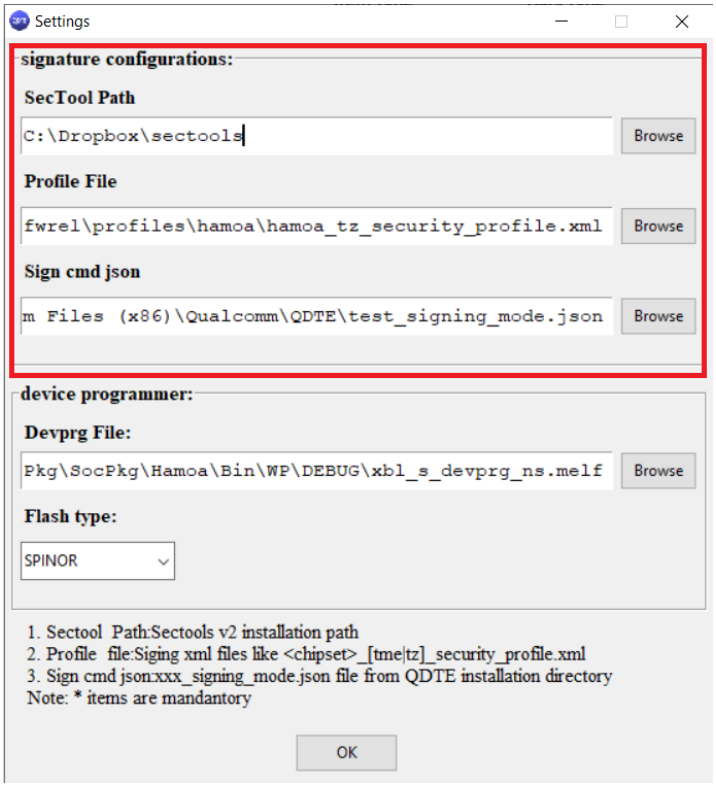
Reassemble unsigned DTB ELF files

Click “Reassemble” button on DTB ELF helper window, it will prompt windows for selecting saving location and desired output filename. It will only reassemble DTB elf and not sign it.



Reassemble and sign DTB ELF file

For reassemble and sign DTB ELF file, Sectool path, Profile file and Sign cmd json need to be configured in settings menu.



Click “Reassemble and Sign…” button on DTB ELF helper window, it will prompt a window for selecting location to save generated DTB elf file. Note output DTB elf file can’t overwrite old DTB elf file. After this, clicking ‘OK’ button. QDTE will start reassembling and signing DTB elf. Once it is successful, “Reassembling and Signing Success!” will be appeared in the console window.

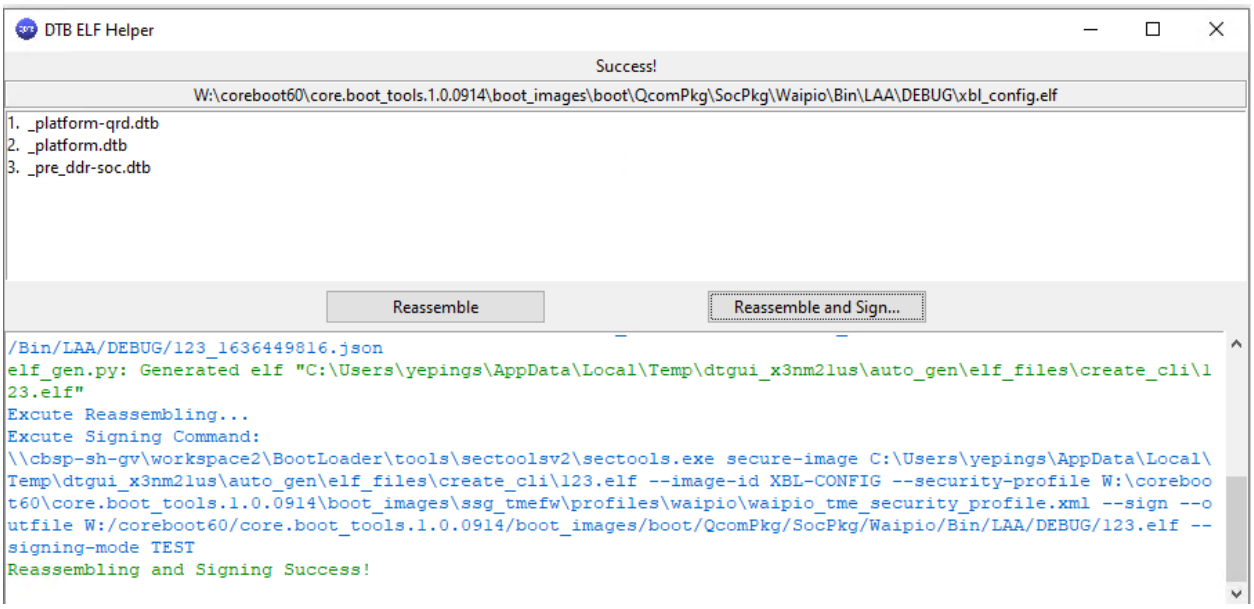


Figure 7: A sample of successfully reassemble and sign DTB Elf file.

Change Reports

A change report is a JSON file containing human- and machine-readable information about changes that have been made to the DeviceTree. The structure of the JSON file is an array of objects. Each object describes a single modification made and will always have the `operation` key to describe the operation type, which will affect the other keys in the object. One other key that is ubiqitous to all operations, although not always present, is the `hash` key, which stores the SHA256 hash of the DTB **after** each operation is performed. This key can be used to verify that the correct file is being modified, or that modifications are being correctly performed to the DeviceTree.

```
{
  "operation": "ADD_PROPERTY",
  "hash": "91c38977f81df5beab2dfe22523ac62fb13bf0a2f9130ea2e9fb80818b9c9348",
  "path": "/memory/temp",
  "name": "test",
  "type": "STRINGS",
  "value": [
    "test with values"
  ]
},
```

Figure 8: A sample single change in a change report. This change describes the addition of a property (ADD_PROPERTY) to the DeviceTree. The name of the new property is `test`, the path to its parent is `/memory/temp`, and it is a `STRINGS` property with a single value of "`test with values`". After applying the operation, the SHA256 hash of the DTB was `91c38977f81df5beab2dfe22523ac62fb13bf0a2f9130ea2e9fb80818b9c9348`.

Operation Types

There are a number of different operation types that can appear in a change report. Besides the common keys of `operation` and `hash`, the following are the different operation types and the keys that they must have:

NULL

This operation does nothing and requires no keys.

LOAD

This operation describes the loading of a new file to edit and should be the first line in a change report. It must contain the key `filename`, which will have the value pointing to the path to the file that was opened.

EDIT_PROPERTY_VALUE

This operation describes the editing of a property value. It must contain the key `path`, which will have the path to the property that was modified, and `value`, which will contain the new value of the property.

ADD_NODE

This operation describes the addition of a new node. It must contain the key `path`, which will have the path to the parent of the new node, and `name`, which will contain the name of the newly added node.

ADD_PROPERTY

This operation describes the addition of a new property. It must contain the key `path`, which will have the path to the parent of the new property, `name`, which will contain the name of the newly added property, and `type`, which will contain the type of the newly added property (options are EMPTY, STRINGS, BYTES, and WORDS). Additionally, if the newly added property type is not EMPTY, the `value` key must be specified, which will contain the value of the new property.

DELETE_NODE

This operation describes the deletion of a node (and all of its children) from the DeviceTree. It must contain the key `path`, which will have the path to the parent of the removed node, and `name`, which will contain the name of the removed node itself.

DELETE_PROPERTY

This operation describes the deletion of a property from the DeviceTree. It must contain the key `path`, which will have the path to the parent of the removed property, and `name`, which will contain the name of the removed property itself.

RENAME_PROPERTY

This operation describes a rename operation on an existing property. It must contain the key `path`, which will have the path to the parent of the property, `to`, which will contain the new name of the property, and `from`, which will contain the old name of the property.

Change Reports with DTB ELF

When exporting an DTB ELF, a change report will automatically be saved. These change reports are slightly modified, since they need to describe the changes made to potentially multiple files. The

modified format contains a top-level JSON object whose keys are filenames within the DTB ELF file, and the values corresponding to the keys are arrays containing the change reports of each file. One important difference is, however, that the LOAD operation does not appear in this change report, since it is not necessary to describe what file is being opened, given that the key already contains this information.


```
{
  "_dloader.dtb": [
    {
      "operation": "EDIT_PROPERTY_VALUE",
      "hash": "2abf974883d1bda3087b265b27684ea47de40ec753b73c8b807fe26e0703c08e",
      "path": "/d",
      "value": [
        3735928493,
        4294967295
      ]
    }
  ]
}
```


Figure 9: An example of the modified change report format for DTB ELF file. Note the lack of a LOAD operation, and the way that multiple changed files could be supported by adding more keys to the top-level object.