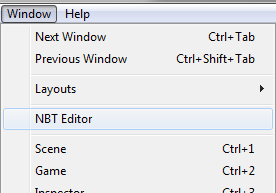
Library Documentation

Phaser Configurator

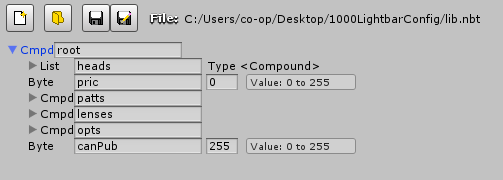
The Library File (as well as the various Bar Files, including the presets) is a binary file formatted in the Named Binary Tag (or NBT) format. In short, the files consist of several chunks of data called Tags that each have their own name to aid in parsing. Most types of Tag contain a single value[[1]](#footnote-1), but a few allow a Tag to contain multiple values[[2]](#footnote-2). All NBT Files consist of a single Tag, typically a Compound Tag so they can hold more than a single piece of information. For more in-depth reading, the technical documentation is available online at:

[http://web.archive.org/web/20110723210920/http://www.minecraft.net/docs/NBT.txt](http://web.archive.org/web/20110723210920/http:/www.minecraft.net/docs/NBT.txt)

Because NBT files are binary files, you’ll require a separate program that can read and modify them. I’ve left behind a Unity extension that works with any NBT file. To access it, click on “NBT Editor” from the Window menu:

The Unity NBT Editor has a few functions on it. You can opt to create a new file from scratch, open an existing one, and save the one you’re currently on (optionally, to another location). These file-related functions are on the top-left of the Editor, next to the path to the file you’re currently editing:

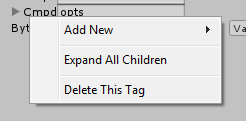
Beneath that, the data contained in the file can be found in the hierarchical format it was parsed in. Compound, List, ByteArray and IntArray tags can be expanded using the foldout arrow on their left.



Each Tag has two components to it: a name and a payload. A name is an identifying UTF-8-encoded string, and is listed immediately after the Tag’s type. The payload is whatever value the Tag is supposed to hold – for Tags which hold multiple values2, the payload is hidden using the foldout. List Tags are special, as they hold an organized list of *unnamed* Tags; for each of the Tags that List Tags contain, their name is assumed to be an empty string and can’t be changed to anything else.

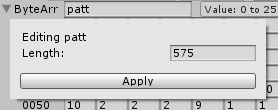
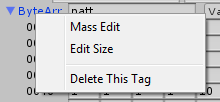
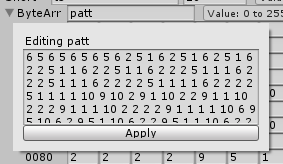
As the above screenshot shows, both the name and the payload of a Tag are exposed. Names and values can be changed easily when possible.

To manipulate the Tags themselves, the far left segment of the Tag (the segment indicating what type of tag it is) can be right-clicked to open a context menu of actions that can be taken on the selected Tag:

All Tags (except for the root tag which is the basis for the entire file) can be deleted. ***This is an immediate action that cannot be undone***, so please take care when selecting this option; if you accidentally delete a Tag, you can reload the whole file via the Open button, as actions to a file aren’t saved until the Save button is clicked.

Compound and List Tags both share the ability to recursively expand their ancestors to aid in finding a specific Tag with as few clicks as possible. They also have the option to add new Tags to themselves via the same context menu – while Compound Tags can add any type of Tag, List Tags are typed to only hold one specific type of Tag and can therefore only add that type. List Tags thus have the option to change what type of Tag it contains. If a List Tag contains Tags when changing it to a different type, the editor will ask whether you want to clear the Tag to perform the change.

IntArray and ByteArray Tags encompass an array of 32-bit integers and 8-bit bytes, respectively. They both have a fixed size, defined with a 32-bit integer, which can be modified via the “Edit Size” action from Tag’s context menu. Additionally, both types of Tags have a “Mass Edit” action, with which a batch of values can be copy-pasted into or out of the Tag.



After the above crash course on how to edit NBT files comes documentation on the format of the library file. The library file has six children off its root: two Byte Tags, a List Tag, and three Compound Tags. (Please be aware that the listing of Tags in the editor is not guaranteed to be in the order listed below – Compound Tags do not preserve the order of tags like List Tags do.)

The pric Byte Tag is a simple flag for the Configurator. The mere existence of this tag tells the Configurator it’s allowed to display prices to the user. If this tag is deleted, any instance of the Configurator reading the file will delete the Display Prices button, preventing the user from turning price display on.

The canPub Byte Tag is a bit field for the Configurator. A high bit in this Tag’s value means the Configurator can create a specific page for the exported PDFs:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Least Significant Bit | | | | | Most Significant Bit | | | |
| 1 | 2 | 3 | 4 | 5 | | 6 | 7 | 8 |
| Summary | BOM | Wiring | Programming | Production Checklist | | Output Usage | reserved | reserved |
|  |  |  |  |  | |  |  |  |

The heads List Tag contains the definitions for every optic the Configurator should offer. It is of type Compound, whose children follow this structure:

* String name – Name of the optic
* String part – The portion of the optic’s part number that doesn’t change from style to style
* Int cost – The sale price of the optic, in whole cents
* Int amp – The current draw of the optic, in while milliamps
* Byte lg – The existence of this Tag tells the Configurator this optic fits into large slots (type doesn’t matter)
* String lgEq – The name of the optic to use when switching from a small optic to a large one (applicable only if small)
* Byte sm – The existence of this Tag tells the Configurator this optic fits into small slots (type doesn’t matter)
* String smEq – The name of the optic to use when switching from a large optic to a small one (applicable only if large)
* List (type Compound) styl – The list of all potential style options for this optic
  + For each element:
    - String name – Name of the style
    - String suff – The portion of the optic’s part number that indicates the optic is this style
    - ByteArray clr – A four-byte array which calls out a specific color in Red-Green-Blue-Alpha format; used for display and color testing against lenses / optics
    - ByteArray clr2 – A four-byte array which calls out a specific color in Red-Green-Blue-Alpha format (applicable only for dual-color optics); used for display and color testing against lenses / optics
* List (type String) locs – The list of all the potential locations this optic can reside, which are denoted as strings detailed thus:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| “f” = front | “fc” = front corner | “a” = alley | “rc” = rear corner | “rf” = far rear | “r” = rear |
|  |  |  |  |  |  |

The lenses Compound Tag contains information about the lens / dome options the Configurator should offer.

* String smFix – The common part number prefix for all center lenses / domes
* String lgFix – The common part number prefix for all end lenses / domes
* List (type Compound) opts – The list of all options for the Configurator to offer
  + For each element:
    - String name – Name of the option
    - String part – The part number suffix of the option
    - ByteArray clr – A four-byte array which calls out a specific color in Red-Green-Blue-Alpha format; used for display and color testing against light heads

The opts Compound Tag contains information about the peripheral options the Configurator should offer. Most of these are offered under the Bar Menu within the application, however some items are not be editable without modifying the application itself.

* Int bracket – The static price for the mounting bracket that’s included with all bars, in whole cents
* Compound base – Information about the base subassemblies
  + Compound part – Contains the model numbers for the base subassembly in use
    - String 0 – Model number for a 37” bar
    - String 1 – Model number for a 44” bar
    - …
  + Compound price – Contains the sale prices for the base subassembly in use
    - Int 0 – Sale price for a 37” bar, in whole cents
    - Int 1 – Sale price for a 44” bar, in whole cents
    - …
* List (type Compound) cableLength – Gives the options for cable lengths. ***Requires Configurator modification to accompany if the length of this List Tag is modified.***
  + For each element:
    - Byte len – How long this option is for, in whole feet
    - Int pwr – Sale price of the power cable in this length
    - Int can – Sale price of the CAN communication cable in this length
    - Int hard – Sale price of the Hardwire communication cable in this length
* List (type Compound) mountingKits – Gives the options for mounting kits. ***Requires Configurator modification to accompany if the length of this List Tag is modified.***
  + For each element:
    - Byte which – Which option in the Configurator this option defines. As of the writing of this document:

|  |  |
| --- | --- |
| 0 = Black Plastic Permanent Mount Kit | 2 = Adjustable Pivot Mount Kit |
| 1 = Pivot Mount Foot Kit | 3 = Towbar Mount Kit |

* + - String part – Part number for this mounting kit
    - Int cost – Sale price of this mounting kit, in whole cents
    - String name – The display name of the mounting kit in the PDF and BOM. ***This value does not change the display name in Bar Menu.***
* Compound cables – Contains information the BOMCables object deals with, including but not limited to cable details
  + String canModPart – The part number for the CAN Breakout Box module
  + String circuit – The part number prefix for the central control circuit
  + Compound intern – Contains the part number prefixes for the internal cables
    - String long – When using a longer bar, this prefix is used for the *driver* side internal control cables
    - String shrt – When using a shorter bar, this prefix is used for the *driver* side internal control cables; this is also the prefix used for the *passenger* side internal control cables, regardless of length
    - String split – The part number for the output splitter, used when two or more heads need to share an output
  + Compound extern – Contains the part number prefixes for the external cables
    - String power – The part number prefix used for the power cable, if used; the cable length is appended to this
    - String CanPre – The part number prefix used for the CAN communication cable, if used; the cable length is appended to this
    - String HardPre – The part number prefix used for the Hardwire communication cable, if used; the cable length is appended to this
  + Compound prices – Contains pricing information. All prices in this Compound Tag are in units of whole cents.
    - Compound circuit – Prices for the various PCBs
      * Int sing – Price for the single-color capable control circuit
      * Int dual – Price for the dual-color capable control circuit
  + ( Compound prices cont’d )
    - * Int canMod – Price for the CAN module
    - Compound intern – Prices for the internal cables
      * Int singL – Price for the longer internal control cable, single-color version
      * Int singS – Price for the shorter internal control cable, single-color version
      * Int dualL – Price for the longer internal control cable, dual-color version
      * Int dualS – Price for the shorter internal control cable, dual-color version
      * Int split – Price per output splitter cable

The patts Compound Tag contains pattern information for flashing and traffic director heads.

* Short base – How long, in units of 10 milliseconds, does a single tick take for all of the patterns?
* List (type Compound) sflsh – Patterns that only influence one head. Elements can take one of two forms.
  + Type 1 – pure definition. Does not depend on others. For each element of this type:
    - String name – Name of the pattern
    - Short id – ID of this pattern
    - Short t0 – length in ticks of “time zero”
    - Short t1
    - Short t2
    - Short t3
    - ByteArray patt – defines the pattern. Each element of this array should follow this syntax:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Most Significant Bit | | | | Least Significant Bit | | | |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Reserved – not currently used | | | | Which time length to use | | Phase A on | Phase B on |

* + Type 2 – reference definition. Depends on others for definition. For each element of this type:
    - String name – Name of the pattern
    - Short id – ID of this pattern
    - List (type Compound) ref – The other patterns this pattern references. The order of the Tags in this List Tag is important, it will start with the first element and continue to the end, then loop.
      * For each element:
        + Short id – ID of the pattern to use
        + Short cnt – How many times to cycle through the pattern before moving to the next item
* List (type Compound) flash – Patterns that influence multiple heads
  + For each element:
    - String name – Name of the pattern
    - Short id – ID of this pattern
    - Short t0 – length in ticks of “time zero”
    - Short t1
    - Short t2
    - Short t3
    - IntArray patt – defines the pattern. Each element of this array should follow this syntax:

|  |  |  |  |
| --- | --- | --- | --- |
| Most Significant Bit | Least Significant Bit | | |
| 32 - 16 | | 15 - 14 | 13 - 0 |
| Reserved – not currently used | | Which time length to use | Whether the output is on |

* List (type Compound) dcflash – Patterns that influence both colors on one head
  + For each element:
    - String name – Name of the pattern
    - Short id – ID of this pattern
    - List (type Compound) ref – The other patterns this pattern references. The order of the Tags in this List Tag is important, it will start with the first element and continue to the end, then loop.
* ( List (type Compound) ref continued )
  + - * For each element:
        + Short id – ID of the pattern to use
        + Short cnt – How many times to cycle through the pattern before moving to the next item
        + Byte clr – If 0, the pattern flashes on color 1. If 1, the pattern flashes on color 2.
* List (type Compound) traff – Traffic patterns. Each definition below uses the
  + For each element:
    - String name – Name of the pattern
    - Short id – ID of this pattern
    - Short t0 – length in ticks of “time zero”
    - Short t1
    - Short t2
    - Short t3
    - Compound 6hed – The definitions to use if six heads are being used
      * IntArray left – The definition to use when showing the left directing pattern
      * IntArray rite – The definition to use when showing the right directing pattern
      * IntArray cntr – The definition of the center directing pattern. Not used currently.
    - Compound 8hed – The definitions to use if more than six heads are being used
      * IntArray left – The definition to use when showing the left directing pattern
      * IntArray rite – The definition to use when showing the right directing pattern
      * IntArray cntr – The definition of the center directing pattern. Not used currently.

1. An 8-bit byte, 16-bit short, 32-bit integer, 64-bit long, 32-bit float, 64-bit double, or a UTF-8-encoded string [↑](#footnote-ref-1)
2. an array of bytes, an array of integers, an ordered list of Tags, or an unorganized clump of Tags [↑](#footnote-ref-2)