QuickTab v1.0

(QTab Primer)

What Is QTab?

QTab is a compact, text-based tablature language for guitar and fretted instruments. It encodes notes, chords, tunings, and performance gestures in a machine-readable but musician-friendly format. Built to integrate with QTab Composer and QTab MIDI Builder, it allows a written line of text to become both notation and playback data.

Unlike static tablature, QTab is modular and symbolic:

- A single file can define tunings, reusable phrases, or full pieces.
- The same syntax can generate notation, sound, or visual representations.
- Every element follows a consistent, minimal structure.

Think of QTab as “music as source code” — precise enough for a parser, but expressive enough to describe human performance.

Example:

```
T[std]: 1:E,2:B,3:G,4:D,5:A,6:E
P[riff]: 6:3q,5:2q,4:0h
```

This defines a standard tuning, T[std], and a short phrase P[riff] where:

- String 6, fret 3 plays a quarter note.
- String 5, fret 2 follows as another quarter note.
- String 4, fret 0 includes a hammer-on (h) technique.

QTab focuses on readability, rhythm integration, and algorithmic structure — connecting notation, performance, and sound synthesis in one consistent grammar.

Foundational Concepts

Every QTab file is made up of declarative building blocks. Each block defines or references a musical object.

Core Object Types

- T = Tuning definition

```
`T[std]: 1:E,2:B,3:G,4:D,5:A,6:E`
```

- C = Chord definition

```
`C[Gmaj]: 1:3,2:0,3:0,4:0,5:2,6:3`
```

- P = Phrase or pattern

```
`P[intro]: 6:3q,5:2q,4:0h`
```

Syntax at a Glance

- Numbers represent strings, numbered from high E (1) to low E (6).
- Colons separate string numbers from fret values.
- Durations are appended as symbols (`w h q e s t`) for whole → thirty-second.
- Techniques follow fret or duration tokens (e.g., `3h`, `5p`, `7/`).
- Brackets `[ ]` group notes to be played simultaneously, like chord stacks.
- Commas `,` separate events happening one after another.
- Pipes `|` separate sequential phrases or chords, creating a timeline.

Example:

```
6:3q,5:2q,[4:3,3:2]q
```

This phrase describes:

1. A note at string 6, fret 3 (quarter note).
2. Then string 5, fret 2 (quarter note).
3. Then a simultaneous double-stop, frets 3 and 2 on strings 4 and 3, respectively.

QTab's structure allows you to think in time-sequenced events or stacked polyphony with equal fluency. Each symbol in the syntax corresponds to a logical event that the parser can transform into musical data — applied visually (tab) or audibly (MIDI).

Tuning

Tunings in QTab define the open-string pitches for your instrument.

They serve as a reusable reference anywhere in your QTab document — so a tuning only needs to be declared once.

Each tuning definition begins with a T[…] block:

```
T[name]: 1:note,2:note,3:note,4:note,5:note,6:note
```

Basic Structure

`T[name]`	Defines the tuning name for later reference
`1:note`–`6:note`	Maps each string number to its open note
Numbers	Always run from 1 (high E) → 6 (low E)
Notes	Use standard note tokens (E, F#, Bb, etc.)

Example:

```
T[name]: 1:E,2:B,3:G,4:D,5:A,6:E
```

This defines T[std] — standard guitar tuning, with open notes from high to low E.

Alternate Tunings

Tunings can easily define alternate setups:

```
T[dropD]: 1:E,2:B,3:G,4:D,5:A,6:D
T[DADGAD]: 1:D,2:A,3:G,4:D,5:A,6:D
T[openG]: 1:D,2:B,3:G,4:D,5:G,6:D
```

Once declared, these names can be recalled anywhere:

```
T[dropD]
P[riff]: 6:0q,6:2q,6:4h
```

This sets the current tuning environment to Drop D, then plays a short phrase in that tuning.

Tip:

Because tunings are discrete definitions, composers can quickly switch tuning references mid-composition. A single QTab file could include multiple movements with different tunings, all under one syntactic system.

Notes:

- Order of strings can be either direction (1→6 or 6→1).
- Commas separate string definitions for clarity.
- Values are string:number pairs where the number indexes fret assignments later.
- Named tunings can be reused anywhere.

Phrases

A Phrase in QTab represents a playable sequence of notes, double-stops, or chords — the building blocks of musical ideas.

Each phrase definition begins with a P[…] block and specifies which string and fret are played, along with optional durations and techniques.

```
P[name]: string:fret[d][t], ...
```

- `string` — which string number to play (1–6).
- `fret` — which fret number, `0` for open, `x` or `X` for muted.
- `d` — (optional) duration (`w h q e s t ...`).
- `t` — (optional) technique (`h`, `p`, `/`, `\`, `b`, etc.).
- `,` separates sequential events (in time).
- `[ ... ]` groups notes to occur simultaneously.

Examples:

Linear Phrase (single-note run):

```
P[line]: 6:3q,5:2q,4:0h
```

Plays string 6 fret 3 (quarter), then 5th string fret 2 (quarter), then 4th string open with a hammer-on.

Simultaneous Notes (double-stop):

```
P[double]: [4:3,3:2]q
```

Plays strings 4 and 3 together on one quarter beat.

Combined Example:

```
P[riff]: 6:3q,5:2q,[4:3,3:2]q,6:1q
```

Translates musically to three single notes followed by a double-stop, then another single low note.

Example with Duration Variations:

```
P[mix]: 6:3h,5:2q.,5:3e,4:0s
```

Here, each token carries its own time value — a half note, a dotted quarter, an eighth, and a sixteenth, respectively.

Phrase Using Techniques:

```
P[slide]: 6:3e/,5:5q,4:0e\.
```

Sequence showing a slide up, a sustained note, and a slide back, with precise timing.

Grouping and Timing Details

- A comma `,` separates notes that happen in sequence.

- A bracket `[ ]` groups notes that occur in parallel (same beat).
- Duration markers (`q`, `e`, etc.) are local — each event carries its own duration unless shared by a bracketed group.
- Complex phrasing (like arpeggios or syncopated runs) can be composed by chaining multiple phrase blocks with the `|` separator (covered in later sections).

Example with time grouping:

```
P[chord_run]: [6:3,5:2,4:0]q,[5:3,4:2]e,6:1h
```

That's a full quarter-note chord hit, a follow-up eighth-note double-stop, and a hammered bass note.

Notes:

- Strings: Numbered 1 (high E) → 6 (low E).
- Frets: Use integers (0=open, x=dead, 12+ allowed).
- Durations:
- Comma `,`: Note and Chord delimiter.
- Simultaneity Grouping [ ]: Stack notes vertically on the same beat (a chord)

This allows phrases to combine single-note runs, double-stops, or grouped voicings.

Durations & Time

Rhythm in QTab is represented by duration symbols appended to fret or group definitions. They define how long each note, chord, or phrase event lasts relative to a beat.

Duration Symbols

Durations are lowercase for readability and follow standard rhythmic values:

Symbol	Name	Relative Value
w	Whole	4 beats
h	Half	2 beats
q	Quarter	1 beat
e	Eighth	½ beat
s	Sixteenth	¼ beat
t	Thirty-second	⅛ beat

Each duration symbol can be extended using dots for rhythmic lengthening:

‘.’ Adds half of the note's value: `q.` = dotted quarter (1.5 beats)
‘..’ Adds 3/4 of the note's value: `e..` = double-dotted eighth (7/8 beat)
‘…‘ Adds 7/8 of the note's value: `h...` = triple-dotted half note

Example Phrase with Durations

```
P[dur_test]: 6:3q.,5:2e,5:3e.,4:0h
```

Breakdown:

- String 6 fret 3 — dotted quarter note.
- String 5 fret 2 — eighth note.
- String 5 fret 3 — dotted eighth.
- String 4 open — half note.

Every note can declare its own duration, allowing complex rhythms in one phrase definition.

Shared Durations Inside Groups

If a duration follows a bracketed group, it applies to all notes in that group.

```
[6:3,5:2,4:0]q
```

All three notes play simultaneously as a quarter-note chord.

Combining Durations With Technique Markers

Durations can coexist with hammer-ons, pull-offs, or slides seamlessly:

```
6:3q.h,5:2e.p,4:0h.
```

Here, the rhythm and technique parsing are layered — the timing still maps cleanly under performance logic.

Techniques

Performance gestures in QTab are indicated using technique markers, appended directly after fret values or durations.

They allow QTab to encode expressive articulations like slides, bends, or hammer-ons.

Common Technique Markers:

```
h = hammer-on

p = pull-off

/ = slide Up

\ = slide Back

b = bend
```

Basic Examples

Hammer-ons and Pull-offs

```
P[legato]: 5:3h,5:5h,5:3p
```

Three legato-linked notes: a hammered 3→5 movement and a pull-off back to 3.

Slides

```
P[slide_up]: 6:3/,6:5q,6:7q
```

Smooth slide up the string, sustaining each note by quarter duration.

Bends

```
P[bendline]: 3:5b,3:7q,3:5p
```

Bend on fret 5, then release (via pull-off) after a sustained tone.

Combined Gestures

```
P[smooth]: 5:2e.h,5:5q/,4:3e\.
```

A hammer-on, followed by an upward slide, then a downward slide back.

Timing & Expression Context

Techniques do not alter duration unless a duration symbol is specified.

For example:

```
5:3p (default time context)
5:3pe (explicit eighth-note pull-off)
```

Each event in QTab can carry multiple modifiers to reflect performance detail.

This design keeps the core grammar lightweight while still representing expressive play styles found in guitar notation and MIDI articulation.

Chords

Chords in QTab define fixed voicings across strings.
They can be used inline, referenced by name, or combined with techniques and arpeggiation patterns in sequences.

Basic Syntax

A chord definition begins with the C[…] symbol:

```
C[name]: 1:fret,2:fret,3:fret,4:fret,5:fret,6:fret
```

Each string number is assigned a fret value (or `x` for muted).

C[name]	Chord identifier name
1:0	High E (open)
X/ X	Muted or unused string
0-22	Fret numbers (standard range)

Basic Example

```
C[Aopen]: 1:0,2:2,3:2,4:2,5:0,6:0
```

Multiple chord definitions can appear in the same document and be invoked by reference.
Partial & Simplified Chords

You may define just the strings used:

```
C[Gtriad]: 3:0,4:0,5:2
```

If your parser supports optional partial definitions, other strings will remain silent or muted by default.

Inline References

Referencing chords is simple:

```
C[Aopen] | C[Gchunk] | C[Djang]
```

Each chord is played in sequence (one per time position).

You can also mix literal voicings and references

```
C[Aopen] | [6:3,5:2,4:0]q | C[Djang]
```

Combines a named chord, an ad-hoc voicing, and another named chord for variety.

Arpeggiation

Arpeggiation modifies how a chord is played — not what strings or pitches it contains.

Use directional markers placed before the chord reference:

- `>` Upward: Plays lowest to highest
- `<` Downwards: Play highest to lowest

These markers are lightweight modifiers and can be combined sequentially within phrases or performance chains.

Advanced Use: Embedding Chords in Phrases

You can treat chords as phrase events:

```
P[intro]: C[Gchunk],C[Djang]
```

or combine them with durations and ornamentation:

```
P[roll]: >C[Aopen]q,>C[Gchunk]q,<C[Djang]h
```

Each acted chord carries both rhythmic and directional information — fully compatible with QTab's timeline.

Sequences

Sequences create timelines — the top-level flow of musical events.
They link chords, phrases, riffs, and tunings into cohesive performances.

QTab sequences use the pipe (`|`) as a structural separator for chronological order.

Syntax Overview

```
element1 | element2 | element3 ...
```

Each element can be:

- A chord reference (`C[name]`)
- A phrase reference (`P[name]`)
- A raw phrase body (`6:3q,5:2q`)
- An arpeggiated element (`>C[Gchunk]`)
- Even a tuning or state change (`T[dropD]`)

Example

```
T[std]
C[Gmaj]: 1:3,2:0,3:0,4:0,5:2,6:3
C[Cmaj]: 1:0,2:1,3:0,4:2,5:3,6:x
P[riff]: 6:3q,5:2q,[4:3,3:2]q

C[Gmaj] | P[riff] | C[Cmaj]
```

This describes three timeline events:

1. Play a G major chord.
2. Execute the riff phrase.
3. Follow with a C major chord.

All conform to the same QTab grammar, so the parser can tokenize them seamlessly.

Mixed Sequences

Sequences can blend directly encoded phrases with defined references:

```
C[Gmaj] | 6:3q,5:2e,4:0e | C[Cmaj] | [4:3,3:2]q
```

This structure alternates literal phrasing (`6:3q,5:2e,4:0e`) between defined chords.

Repetition & Variants

You can define sequence snippets and reuse them:

```
P[fill]: 6:3q,6:5e.,6:7e
Seq[chorus]: C[Gmaj]|P[fill]|C[Cmaj]
```

Here a reusable phrase P[fill] is embedded within a longer sequence Seq[chorus] — a modular structure ideal for composition.

Example: Complete Performance Chain

```
T[std]
C[Gmaj]: 1:3,2:0,3:0,4:0,5:2,6:3
C[Dmaj]: 1:2,2:3,3:2,4:0,5:x,6:x
P[riff]: 6:3q,5:2e,5:3e,[4:3,3:2]q

Sequence:
C[Gmaj] | P[riff] | C[Dmaj] | P[riff]
```

This represents a repeatable verse structure with two chords and a riff looped between them.

From this, QTab Composer can generate notation, tab, or synchronized MIDI playback — all from a single unified representation.

File Format & Parser Implementation

.qtab File Structure

A .qtab file is a plain text document that can contain any combination of definitions, references, and sequences.

Files are human-readable and structured for easy parsing by tools like QTab MIDI Builder and QTab Composer.

Basic .qtab Template:

```
# Optional comment header (lines starting with #)
T[tuning_name]: 1:note,2:note,3:note,4:note,5:note,6:note
C[chord_name]: 1:fret,2:fret,3:fret,4:fret,5:fret,6:fret
P[phrase_name]: phrase_body

# Main performance sequence
C[chord1] | P[phrase1] | 6:3q,5:2q | C[chord2]
```

QTab MIDI Builder Parser Behavior

QTab MIDI Builder implements practical shortcuts beyond the strict grammar for real-world tablature writing:

Partial Chords	All 6 strings required
Mute Notation	× or X
Inline Chords	CInamel prefix
Default Duration	Explicit required
Case Sensitivity	E, F# exact
String Order	1→6 only

Example qtab File

```
# Drop D Blues Progression – QTab v1.0
T[dropD]: 1:E,2:B,3:G,4:D,5:A,6:D

# Roman numeral chord voicings
C[I]: 6:0,5:0,4:2,3:2,2:0,1:0
C[IV]: 6:5,5:5,4:7,3:7,2:5,1:x
C[V]: 6:3,5:5,4:5,3:5,2:3,1:x

# Riff phrase (pentatonic position)
P[blues_riff]: 6:0q,6:3q,5:0e,5:2e,5:3q,[4:2,3:0]h

# Main 12-bar sequence
C[I] | P[blues_riff] | C[IV] | C[V] | C[I]
```

This single file defines a complete Drop D blues progression with chord voicings, a reusable riff, and a timeline sequence — instantly playable via MIDI export or notation rendering.

Parser Processing Order

1. Tuning declarations set the pitch reference for all subsequent elements
2. Chord/Phrase definitions (`C[]`, `P[]`) are stored for reference
3. Sequence elements (`|` separated) execute chronologically
4. Inline phrases render immediately within their time position

```
T[dropD]        # Sets global tuning context
C[I]            # Chord stored, not played yet
P[riff]         # Phrase stored, not played yet
C[I] | P[riff]  # NOW both execute in sequence
```

File Extensions & Tooling

.qtab	Source notation (Tools: QTab Composer, MIDI Builder)
.qtab .json	Parsed JSON export (Tools: DAWs, MIDI sequencers)
.mid	MIDI playback (Tools: Any MIDI player)

Pro Tip: Save your working files as `.qtab` for editing, then export to `.mid` or `.qtab.json` for integration with other music software.

EBNF Grammar Summary

The formal grammar of QTab v1.0, expressed in Extended Backus-Naur Form (EBNF). This defines exactly what strings are valid QTab syntax.

```
(* QTab v1.0 Core Grammar *)

qtab_doc     = { definition | sequence }* ;

definition   = tuning_def | chord_def | phrase_def ;

sequence     = element , { "|" , element } ;


(* Named Definitions *)

tuning_def   = "T[" , name , "]:" , string_map ;

chord_def    = "C[" , name , "]:" , string_frets ;

phrase_def   = "P[" , name , "]:" , phrase_body ;


(* References *)

tuning_ref   = "T[" , name , "]" ;

chord_ref    = "C[" , name , "]" ;

phrase_ref   = "P[" , name , "]" ;


(* Core Elements *)

element      = [ arpeggio_mark ] , ( chord_ref | phrase_ref |
phrase_body ) ;

phrase_body  = phrase_event , { "," , phrase_event } ;

phrase_event = single_note | simultaneous_group ;


(* Note Structure *)

single_note     = string_num , ":" , fret_value , [ duration ] ,
[ technique ]* ;
```

```
simultaneous_group = "[", single_note , { "," , single_note } , "]" ,
[ duration ] ;


(* Structural Components *)

string_map    = string_assign , { "," , string_assign } ;

string_assign = string_num , ":" , note_name ;

string_frets  = string_note , { "," , string_note } ;

string_note   = string_num , ":" , fret_value ;


(* Atomic Values *)

fret_value    = "x" | "X" | "0" | fret_number ;

fret_number   = "0" | nonzero_digit , { digit } ;

duration      = base_dur , [ dot_seq ] ;

base_dur      = "w" | "h" | "q" | "e" | "s" | "t" ;

dot_seq       = "." | ".." | "..." ;

technique     = "h" | "p" | "/" | "\\" | "b" ;

arpeggio_mark = ">" | "<" ;


(* Terminals *)

string_num = "1" | "2" | "3" | "4" | "5" | "6" ;

name       = letter , { letter | digit | "_" } ;

note_name  = ? musical note token ? ;
```

Key Design Principles:

- Single-pass parsing — no lookahead required beyond name resolution
- Unambiguous delimiters — `, | [ ] :` have distinct roles
- Extensible — new techniques/durations added as terminals

Practical Notes & Usage

Writing Effective QTab

Start Simple, Build Modularly:

```
# Good: Progressive complexity
T[std]: 1:E,2:B,3:G,4:D,5:A,6:E
C[Am]: 1:0,2:1,3:2,4:2,5:0,6:x
P[motif]: 5:0q,5:3q,5:5q
C[Am] | P[motif]
```

Common Patterns:

Chord Progression `C[Am] | C[F] | C[G]` Song structure
Arpeggio `>C[Am]q | <C[F]q` Harp-like patterns
Riff `P[riff]: 6:3e,6:5e,6:3e` Repeating phrases
Scale Run `P[scale]: 5:0q,5:2q,5:3q,5:5q` Technical passages

Parser Compatibility Matrix

Troubleshooting Common Issues

“Parser ignores my chord”

❌ C[Am]: 1:0,2:1,3:2 (* incomplete *)

✅ C[Am]: 1:0,2:1,3:2,4:2,5:0,6:x

“Notes play wrong pitches”

❌ T[std]: 6:E,5:A... (* reverse order *)

✅ T[std]: 1:E,2:B,3:G,4:D,5:A,6:E

“Duration feels off”

❌ 6:3,5:2 (* defaults to q *)

✅ 6:3q,5:2e (* explicit timing *)

Toolchain Workflow

.qtab → QTab MIDI Builder → .mid / .qtab.json
→ QTab Composer → SVG tab / notation
→ DAW → Final mix

Next Steps

1. Write your first .qtab using the examples above
2. Test in QTab MIDI Builder — instant feedback loop
3. Build a library of `T[]`, `C[]`, `P[]` definitions
4. Share on GitHub — contribute to the QTab ecosystem

Development Roadmap

- QTab Composer
- QTab MIDI Builder and Player
- MIDI: Steps to Integration/DAW
- DrawBot Integration
- Graphic TAB parser (Python + DrawBot)
- Fretboard SVG renderer

Test it!

Share licks, report issues, open PRs.
By Dennis M. Walsak — [modularmedia.com/qtab]
License: MIT

QTab v 1.0 Syntax Specification (EBNF)

(Extended Backus–Naur Form) defines the exact syntax rules for a language or notation like QTab. EBNF precisely describes what strings are valid (parsable) and what structures they form. QTab parser uses it as a blueprint to tokenize a string like `6:3q,5:2e` into playable notes. QTab EBNF ensures that anyone (or any script) can write `C[Djang]: 1:2,2:3,3:2,4:0,5:x,6:x` correctly, forever.

```
qtab         = { definition | reference } ;

definition   = tuning_def | chord_def | phrase_def ;

tuning_def   = "T[" , name , "]:" , string_map ;

chord_def    = "C[" , name , "]:" , string_frets ;

phrase_def   = "P[" , name , "]:" , phrase_body ;

reference    = tuning_ref | chord_ref | phrase_ref ;

tuning_ref   = "T[" , name , "]" ;

chord_ref    = "C[" , name , "]" ;

phrase_ref   = "P[" , name , "]" ;

qtab_doc     = { definition | sequence } ;

sequence     = element , { "|" , element } ;

element      = chord_ref | phrase_ref | phrase_body | arpeggio_element ;

string_map   = string_assign , { "," , string_assign } ;

string_assign = string_num , ":" , note_name ;

note_name    = ? valid note token, e.g. E, Eb, F#, etc. ? ;

string_frets = string_note , { "," , string_note } ;

string_note  = string_num , ":" , fret_value ;

fret_value   = ( fret_number | "x" | "X" ) ;

fret_number  = nonzero_digit , { digit } | "0" ;  (* open string allowed
*)

phrase_body  = phrase_string , { "/" , phrase_string } ;

phrase_string = simultaneous_group | single_string ;
```

```
simultaneous_group = "[", string_group , "]" ;  (* vertical stack, same
beat *)

single_string     = string_num , ":" , note_unit , { "," , note_unit } ;

string_group      = string_num , ":" , note_unit , { "/" , string_num ,
":" , note_unit } ;

note_unit    = fret_value , [ duration ] , [ technique ] ;

duration     = "w" | "h" | "q" | "e" | "s" | "t" , [ "." ] ;

technique    = "h" | "p" | "sl" , [ fret_number ] | "b" ;

string_num   = "1" | "2" | "3" | "4" | "5" | "6" ;

name         = letter , { letter | digit | "_" } ;

arpeggio_element = [ arpeggio_mark ] , element ;

arpeggio_mark = ">" | "<" ;

separator     = "|" ; (* separates chords and phrases *)



(* Examples:

   T[std]: 1:E,2:B,3:G,4:D,5:A,6:E

   C[Aopen]: 1:0,2:2,3:2,4:2,5:0,6:0

   [Galt1]: 6:3/5:2,3,5/4:3,5/3:4,5/2:3,6/1:3

   p[riff]: 6:0/[5:2/4:2]/6:0/[5:2/4:2]  → sequential | simultaneous |
sequential

   Aopen|5:2,3|4:0|Gchunk → sequence of chord_ref|phrase_body|phrase_body|
chord_ref

*)
```

Revisions