WiseSeer™

HVAC Enthalpy Calculator

Programmer's Reference Guide

Gulf Coast Edition · Version 1.0

Mobile-First PWA · Vanilla JS · ASHRAE Corrected Data

1. Application Overview

WiseSeer™ is a professional-grade, browser-based Progressive Web App (PWA) built for HVAC field technicians. It calculates total cooling capacity in BTU/hr and tonnage using the enthalpy method, accounting for real-world air density variation with elevation. The app requires no backend, no build step, and no dependencies — it ships as three files and works fully offline after the first load.

1.1 Design Goals

- Mobile-first: designed for one-hand use on a phone at a job site
- High-contrast dark theme (#0A0C0B background, #00C896 accent) for outdoor readability
- Voice I/O: speech recognition for hands-free data entry, speech synthesis for result readback
- Offline-capable: service worker caches all assets on first visit
- No framework, no build tools — pure HTML/CSS/JS in a single index.html
- ASHRAE-corrected enthalpy table with documented data-quality fixes

1.2 File Structure

index.html	`Application shell, styles, logic — entire app in one file`
manifest.json	`PWA manifest: name, icons, theme, display mode`
sw.js	`Service worker: install, activate, fetch with cache-first strategy`

1.3 Browser Compatibility

The app targets Chromium-based mobile browsers (Chrome, Edge, Samsung Internet) and Safari/WebKit on iOS. Voice input uses the Web Speech API (SpeechRecognition), which is fully supported in Chrome/Edge and partially in Safari. Voice output uses SpeechSynthesis, which is universally supported. Graceful degradation hides mic buttons when the API or hardware is unavailable.

2. Physics & Engineering Basis

The enthalpy method is the industry standard for calculating total (sensible + latent) cooling load on a coil. It accounts for the energy in both the temperature and moisture content of air, unlike the simple sensible-only formula (1.08 × CFM × ΔT) which ignores humidity.

2.1 Core Formula

$$\text{Total Cooling (BTU/hr)} = \text{CFM} \times \rho \times 60 \times \Delta H$$

CFM	Airflow in cubic feet per minute (measured or specified)
ρ	Air density in lb/ft³ (elevation-corrected)
60	Converts per-minute to per-hour
ΔH	Return enthalpy minus supply enthalpy, BTU/lb dry air

2.2 Enthalpy of Moist Air

Enthalpy (H) at a given wet-bulb temperature is tabulated from ASHRAE Fundamentals data. Wet-bulb temperature is used because it encodes both sensible temperature and moisture content in a single measurable value — it is what a psychrometric sling thermometer reads after evaporative equilibrium.

The formula behind the table is approximately: $H \approx 0.240 \cdot T + W \cdot (1061 + 0.444 \cdot T)$, where T is dry-bulb temperature (°F) and W is humidity ratio (lb water/lb dry air). The app bypasses this formula and uses the pre-computed, corrected lookup table for speed and accuracy.

2.3 Elevation Correction

Atmospheric pressure decreases with altitude, reducing air density. The sensible heat factor (SHF) used in simplified formulas (1.08 at sea level) also decreases proportionally. At 5,000 ft elevation, air density is roughly 17% lower than sea level, meaning the same CFM carries significantly less energy.

WiseSeer uses a 70-row elevation lookup table covering 0–10,000 ft, providing both SHF and density (lb/ft³) at each step. Values between table entries are linearly interpolated.

2.4 Weight of Air Flow

$$\text{Weight (lb/hr)} = \text{CFM} \times \text{density (lb/ft}^3\text{)} \times 60 \text{ min/hr}$$

This converts the volumetric flow (CFM) into mass flow (lb/hr), which is then multiplied by the enthalpy difference to yield BTU/hr. The density term is what makes elevation correction meaningful.

2.5 Tons of Refrigeration

$$\text{Tons} = \text{BTU/hr} \div 12{,}000$$

One ton of refrigeration equals 12,000 BTU/hr — the rate at which a one-ton block of ice melts over 24 hours (288,000 BTU ÷ 24). This is the standard commercial sizing unit for HVAC equipment.

3. Data Tables & Lookup Logic

3.1 Enthalpy Table Structure

The enthalpy lookup is split into two arrays for precision and storage efficiency:

H_LO	45.0–79.9°F — 350 values, 0.1°F resolution, hand-verified from ASHRAE source
H_HI	80.0–95.0°F — 151 values, generated by linear interpolation between 16 ASHRAE anchor points
H_ANCHORS	16 integer-degree anchor values from ASHRAE (80–95°F whole degrees only)

3.2 Known Data Corrections in H_LO

Four transcription errors were identified and corrected in the source data. These are documented inline in the source code with before/after values:

Wet Bulb (°F)	Original	Corrected	Error Type
59.4	36.05	26.05	Digit transposition (3↔2)
66.7	31.18	31.38	Digit transposition (1↔3)
72.8	35.56	36.56	Dropped leading digit
77.2	40.88	40.78	Digit transposition (8↔7)

3.3 getEnthalpy(wb) — Lookup Function

```
function getEnthalpy(wb) {
  // wb: wet-bulb temperature in °F, range 45–95
  const tbl  = wb < 80 ? H_LO : H_HI;
  const base = wb < 80 ? 45   : 80;
  const raw  = (wb - base) * 10;       // float index
  const lo   = Math.floor(raw);         // lower table index
  const hi   = Math.min(lo + 1, tbl.length - 1);
  return tbl[lo] + (raw - lo) * (tbl[hi] - tbl[lo]); // lerp
}
```

Linear interpolation between adjacent 0.1°F table entries gives sub-0.1°F precision without a larger table. The function returns null for inputs outside 45–95°F.

3.4 Elevation Table Structure

ELEV_DATA is a 70-row array of [elevation_ft, SHF, density_lb_ft3] triplets. Steps are 100 ft from 0–3,000 ft, then 200 ft from 3,000–10,000 ft (coarser spacing where the gradient is shallower). Two corrections were made:

- 1,200 ft density: 0.0818 → 0.0718 (digit transposition: 8 and 1 swapped)
- 8,800 ft density: 0.0754 → 0.0540 (bad entry; replaced with interpolated ASHRAE value)

3.5 getElevFactors(elev) — Lookup Function

```
function getElevFactors(elev) {
  elev = Math.max(0, Math.min(10000, elev)); // clamp
  // binary search for bracket ...
  const t = (elev - lo[0]) / (hi[0] - lo[0]);
  return {
    sh:      lo[1] + t * (hi[1] - lo[1]),   // SHF
    density: lo[2] + t * (hi[2] - lo[2]),   // lb/ft³
  };
}
```

Returns an object with sh (sensible heat factor) and density (lb/ft³). Both are linearly interpolated from the nearest bracket. The SHF is displayed for reference but density drives the BTU calculation.

4. Calculation Engine

4.1 Input Validation

The calculate() function validates all four inputs before proceeding. Validation rules:

Field	Range	Error Behavior
CFM	100 – 10,000	Red border + inline error message; calculation aborted
Elev	0 – 10,000 ft	Defaults to 0 if blank (sea level assumed)
RWB	45 – 95 °F	Red border; matches ASHRAE table range
SWB	45 – 95 °F	Must also be strictly less than RWB

> **NOTE** Supply wet-bulb must be strictly less than return wet-bulb. If swb >= rwb, a specific error is shown on the SWB field: "Must be less than return wet bulb". This is a physics constraint — the coil cannot increase the enthalpy of the air.

4.2 Calculation Steps (in order)

1. Read and parse all four input fields (parseFloat)
2. Validate all fields; abort with field-level errors if any fail
3. Call getElevFactors(elev) to obtain density and SHF
4. Call getEnthalpy(rwb) and getEnthalpy(swb) to get return and supply enthalpy (BTU/lb)
5. Compute ΔH = returnEnthalpy − supplyEnthalpy
6. Compute weightPerHour = CFM × density × 60
7. Compute BTU/hr = weightPerHour × ΔH
8. Compute tons = BTU/hr ÷ 12,000
9. Set extreme flag if rwb > 85°F or swb > 85°F
10. Render all result cells; show/hide warning flag
11. Append entry to job log (localStorage, 10-entry circular buffer)
12. Auto-speak result if voiceOutput setting is enabled
13. Smooth-scroll to results panel on mobile viewports (< 900px)

4.3 Output Fields

res-btu	`BTU/hr — Math.round(), toLocaleString() with commas`
res-tons	`Tons — toFixed(1), one decimal place`
res-rh	`Return enthalpy, toFixed(2) + " BTU/lb"`
res-sh	`Supply enthalpy, toFixed(2) + " BTU/lb"`
res-dh	`Delta enthalpy (ΔH), toFixed(2) + " BTU/lb"`
res-den	`Air density, toFixed(4) + " lb/ft³"`
res-wt	`Weight of air, Math.round() + toLocaleString() + " lb/hr"`
res-sh-factor	`Sensible heat factor, toFixed(3) — from elevation table`

4.4 Extreme Conditions Warning

If either RWB or SWB exceeds 85°F, a prominent orange warning banner appears: "Extreme conditions — wet bulb above 85°F. Gulf Coast severe heat event." This threshold was chosen because conditions above 85°F wet bulb are physiologically dangerous and uncommon in standard HVAC design, warranting technician awareness.

5. Voice Input & Output

5.1 Voice Input Architecture

Voice input uses the Web Speech API (window.SpeechRecognition or webkitSpeechRecognition). Availability is determined at startup by enumerating media devices. Mic buttons remain hidden until both the API is available AND an audio input device is found.

5.2 startMic(fieldId)

Each mic button calls startMic() with its field ID. Only one recognition session may be active at a time — calling startMic() while another is active stops it instead of starting a new one. The function:

- Sets visual state: recording class on button, listening class on input, shows status bar with animated dot
- Creates a new SpeechRecognition instance (non-continuous, up to 3 alternatives)
- On result: iterates alternatives in confidence order, calls wordsToNum() on transcript
- Accepts first alternative meeting the confidence threshold (default 80%)
- Falls back to lowest-confidence alternative if none meet threshold
- Reads back the accepted value via speak() if voiceOutput is enabled

5.3 wordsToNum(str)

Converts spoken natural language numbers to numeric values. Handles:

- Direct numeric strings: "1500", "63.5"
- Word numbers: "fifteen hundred", "sixty-three point five"
- Compound forms: "two thousand five hundred", "one hundred"
- Decimal point: "point" or "decimal" keyword triggers decimal accumulation

> **NOTE** The function first tries parseFloat() on the raw string — if the API transcribes a numeral directly (common for short numbers), word processing is skipped entirely.

5.4 Voice Output

All speech output flows through the speak(text, priority) function. When priority is true, window.speechSynthesis.cancel() is called first to interrupt any in-progress speech. Settings-controlled parameters:

volume	`0–1 (default 0.85)`
rate	`0.7–1.3 (default 0.92 — slightly slower than normal for field noise)`
voice	`voiceURI of selected voice (empty = system default)`

Voice population is triggered by window.speechSynthesis.onvoiceschanged and filtered to English-language voices only. The selected voice URI is persisted in settings.

5.5 Auto-Speak on Calculate

When voiceOutput is enabled, calculating automatically speaks the result in priority mode. The spoken string includes an optional extreme-conditions prefix and the BTU and tonnage values. Example output: "Warning: extreme conditions detected. Total cooling capacity: 48,200 BTU per hour. That is 4.0 tons."

6. Settings System

6.1 Settings Object

All runtime preferences live in a single settings object, loaded from localStorage at startup and merged with defaults using Object.assign():

Key	Default	Description
voiceInput	`true`	Show mic buttons on each input field
voiceOutput	`true`	Speak field readback and result
volume	`0.85`	SpeechSynthesis utterance volume
rate	`0.92`	SpeechSynthesis utterance rate
voice	`""`	voiceURI string; empty = system default
confidence	`0.80`	Minimum SpeechRecognition confidence threshold

6.2 Settings Persistence

saveSettings() serializes the object to JSON and writes to localStorage under the key "wiseseer_settings". applySettings() reads the live settings object (never re-reads localStorage) and synchronizes all UI elements: toggle states, range values, displayed percentages, mic button visibility, and the header VO button icon.

6.3 Settings UI

Settings are presented in a bottom sheet panel (#settings-panel) triggered by the header gear button. The overlay (#settings-overlay) covers the full viewport and dismisses on outside click. The panel slides up via CSS animation (slide-up @keyframes) and respects env(safe-area-inset-bottom) for iOS notched devices.

7. Job Log

The job log provides a persistent, on-device history of up to 10 calculations. It is stored in localStorage under the key "wiseseer_log" as a JSON array of entry objects.

7.1 Log Entry Schema

cfm	`Number — airflow value used`
elev	`Number — elevation in feet`
rwb	`Number — return wet-bulb °F`
swb	`Number — supply wet-bulb °F`
btu	`Number — calculated BTU/hr (integer, pre-rounded)`
tons	`Number — calculated tons (1 decimal, pre-rounded)`
ts	`Number — Unix timestamp in milliseconds (Date.now())`

7.2 Circular Buffer

saveLog() prepends the new entry (log.unshift(entry)), then trims to 10 entries (log.pop() if length > 10). This keeps the most recent 10 calculations with O(1) trim cost. The log is re-read from localStorage on every renderLog() call to ensure freshness.

7.3 Rendering

renderLog() regenerates the #log-entries innerHTML from the log array via .map().join(). Each entry shows a timestamp (locale date + HH:MM time), the four input parameters, and the BTU/tons result. The log is collapsed by default and toggled by the log-header button with a CSS chevron rotation.

8. Progressive Web App (PWA)

8.1 Manifest (manifest.json)

The web app manifest provides metadata for installation on the home screen. Key fields:

display	`"standalone" — hides browser chrome after install`
orientation	`"any" — supports both portrait and landscape`
background_color	`#0A0C0B — splash screen background matches the app`
theme_color	`#0A0C0B — status bar color on Android`
start_url	`"/" — launches from root, enabling offline entry`

The app icon is an inline SVG data URI — no external image file required. The "W" glyph is rendered in monospace at 110px on the dark background.

8.2 Service Worker (sw.js)

The service worker uses a cache-first strategy for all app assets, with a network-first pass-through for anything uncached. Cache name is "wiseseer-v1" — increment the version string to force cache invalidation on deploy.

- install: opens cache, adds /, /index.html, /manifest.json; calls skipWaiting() for immediate activation
- activate: deletes all caches whose key !== CACHE; calls clients.claim() for immediate control
- fetch: Google Fonts URLs get a stale-while-revalidate strategy (cached on first fetch); all other requests use cache-first with network fallback

> **NOTE** To update the app after deployment, change the CACHE constant in sw.js to a new version string (e.g., "wiseseer-v2"). The old cache will be deleted on the next activate event.

9. UI Architecture & CSS Design System

9.1 CSS Custom Properties (Design Tokens)

All colors, radii, and spacing are defined as CSS custom properties on :root. This makes theming a single-point change:

Token	Value	Usage
--bg	`#0A0C0B`	Page background, header background
--surface	`#131614`	Result panel, settings panel, icon buttons
--surface2	`#1A1F1C`	Input fields, mic buttons, result hero
--accent	`#00C896`	Primary CTA, result values, active states
--accent-t	`#00C89620`	Accent at 12% opacity for subtle fills
--warn	`#FF6B35`	Recording state, extreme-conditions flag
--danger	`#FF4545`	Validation errors, clear log button

--tap	`52px`	Minimum touch target height for all interactive elements

9.2 Responsive Layout

Three breakpoints define the layout behavior:

- < 600px (phone): Single column, all sections stacked vertically, 16px gutter padding
- 600–899px (tablet): Two-column field grid, larger result BTU font, settings panel stacked
- ≥ 900px (desktop): Main area becomes flex-row with input-col (340px fixed) and output-col (flex: 1)

9.3 Typography

Two typefaces are used throughout, loaded from Google Fonts:

IBM Plex Mono	`Numeric data: input values, BTU result, enthalpy values, log entries`
Barlow Condensed	`All labels, buttons, settings, navigation, body text`

Both fonts are cached by the service worker on first load for offline availability. Fallbacks are Courier New (mono) and system-ui/sans-serif.

9.4 Keyboard Shortcuts

Shortcut	Action
Tab	Move to next input field
Enter	Trigger calculate() from any input field
Ctrl + M	Start mic on currently focused field (or CFM if no field is focused)
Ctrl + R	Read result aloud (only if results are showing)
Escape	Stop active mic recognition AND cancel any in-progress speech

10. Extending the App

10.1 Adding °C / Metric Support

All internal calculations use °F and imperial units. To add Celsius input, add a unit-toggle UI element and convert inputs before passing to getEnthalpy(): wb_f = (wb_c × 9/5) + 32. Enthalpy output would need BTU/lb → kJ/kg conversion (1 BTU/lb = 2.326 kJ/kg). Air density would shift from lb/ft³ to kg/m³ and CFM to L/s.

10.2 Adding Dry-Bulb + RH Input

Currently the app requires wet-bulb readings. Many digital meters report dry-bulb and relative humidity. The psychrometric relationship between dry-bulb (T), RH, and wet-bulb (TWB) can be approximated with the Stull formula: TWB ≈ T × atan(0.151977 × (RH + 8.313659)^0.5) + atan(T + RH) − atan(RH − 1.676331) + 0.00391838 × RH^1.5 × atan(0.023101 × RH) − 4.686035. Add a mode toggle to switch between WB direct entry and DB+RH entry.

10.3 Adding a Unit System Toggle (SHF-Only Mode)

Some technicians prefer the simplified formula: BTU/hr = 1.08 × CFM × ΔT (sensible only). This could be exposed as an alternate "Sensible Only" mode that takes dry-bulb temperatures instead of wet-bulb, using the SHF from the elevation table instead of full enthalpy math.

10.4 Exporting the Job Log

The log is already serialized JSON in localStorage. To add CSV export: read loadLog(), map entries to CSV rows, create a Blob with type "text/csv", and trigger a download via a temporary anchor element with the download attribute. No server required.

10.5 Updating the Enthalpy Table

H_LO is a hand-verified 350-element array. To replace with updated ASHRAE data, recompute the array at 0.1°F steps from 45.0 to 79.9°F and replace the constant. H_ANCHORS holds 16 whole-degree ASHRAE values (80–95°F); H_HI is auto-generated at startup by interpolating between them. Anchor updates require changing only 16 values.

10.6 Deploying the PWA

- Serve all three files (index.html, manifest.json, sw.js) from the same origin
- Must be served over HTTPS (required for service workers and getUserMedia)
- No build step, no bundler, no npm — static file hosting (GitHub Pages, Netlify, S3+CloudFront) is sufficient
- To force cache refresh after update: increment CACHE version string in sw.js
- iOS Add-to-Home-Screen requires apple-mobile-web-app-capable meta tag (already present)

11. Troubleshooting

Symptom	Likely Cause & Fix
Mic buttons not appearing	No audio input device found, or browser lacks SpeechRecognition API. Chrome/Edge on Android/desktop. Safari on iOS has partial support in iOS 17+.
No speech after Calculate	voiceOutput is off (🔇 icon in header). Or: page was not loaded via user gesture first (required by browser autoplay policy for speech).
App not working offline	Service worker did not register (requires HTTPS). Open DevTools > Application > Service Workers to check. Hard-refresh (Ctrl+Shift+R) does not use SW cache.
Old version showing after deploy	Increment CACHE string in sw.js (e.g., wiseseer-v2). Clear cache in DevTools > Application > Cache Storage.
SWB >= RWB error unexpected	Physics constraint. In cooling mode, supply air must always have lower enthalpy (lower wet-bulb) than return air.
Enthalpy value seems wrong	Check wet-bulb input precision. The table provides ~0.01 BTU/lb resolution. Values outside 45–95°F return null.

Appendix A: Quick Reference Card

Formula / Constant	Value
BTU/hr calculation	`CFM × density × 60 × ΔH`
Tons conversion	`BTU/hr ÷ 12,000`
SHF at sea level	`1.080`
Air density at sea level	`0.0750 lb/ft³`
Enthalpy table range	`45.0 – 95.0 °F wet-bulb`
Extreme condition threshold	`Wet-bulb > 85 °F`
localStorage settings key	`"wiseseer_settings"`
localStorage log key	`"wiseseer_log"`
Max log entries	`10 (circular buffer)`
Service worker cache name	`"wiseseer-v1"`

Appendix B: Data Source Notes

The H_LO array (45–79.9°F) is derived from tabular ASHRAE Fundamentals psychrometric data for moist air enthalpy in imperial units (BTU/lb dry air). The four corrections noted in Section 3.2 were identified by comparing against independently computed psychrometric values using the ASHRAE-RP-1485 equations.

The H_ANCHORS array (80–95°F) uses ASHRAE Fundamentals 2017 Table 2 (Thermodynamic Properties of Moist Air at Standard Atmospheric Pressure) for the whole-degree anchor values. Intermediate 0.1°F values are linearly interpolated and do not represent independent ASHRAE measurements.

The elevation table (ELEV_DATA) is based on the ASHRAE Standard 111 correction factors for air density and sensible heat factor at altitude. The two documented corrections (1,200 ft and 8,800 ft density values) were identified by comparing against the ICAO Standard Atmosphere model.