

Callsign Lookup Algorithm Specification

Version: 0.90.11-Beta Date: 2025-10-06

--- Revision History ---

[0.90.11-Beta] - 2025-10-06

Fixed

- Synchronized documentation with the code baseline in `get_cty.py`.
- Added documentation for the two special-case rules for handling ambiguous KG4 callsigns.

[0.30.43-Beta] - 2025-08-24

Changed

- Corrected Step 6 of the Portable Call Heuristics to accurately describe the code's fall-through logic to the longest prefix match.

[0.30.42-Beta] - 2025-08-15

Changed

- Updated the KG4 special case rule to specify the precise KG4[A-Z]{2} pattern, matching the code's logic.

[0.30.30-Beta] - 2025-08-05

- No functional changes. Synchronizing version numbers.

[0.30.0-Beta] - 2025-08-05

- Initial release of Version 0.30.0-Beta.
- Standardized all project files to a common baseline version.

1. Core Purpose

The script's goal is to replicate the logic of major contest logging programs by implementing a precise, ordered, multi-step algorithm. It takes a raw callsign string as input and returns a data tuple containing the resolved DXCC entity, CQ/ITU Zones, continent, and a portableid field.

2. Output Data Structure

The script's output is a `FullCtyInfo` named tuple, which has been modified to include the `portableid` field. (DXCCName, DXCCPfx, CQZone, ITUZone, Continent, Lat, Lon, Tzone, WAEName, WAEPfx, portableid)

The `portableid` field will contain the specific part of a portable callsign that was used to determine the location (e.g., "7", "VP2V") and will be blank for non-portable callsigns.

3. The Lookup Algorithm

The script follows a strict order of operations. A successful match at any step concludes the algorithm.

Step 1: Pre-processing (`_preprocess_callsign`)

The initial step is to clean the raw callsign string to create a standardized base for analysis. This involves stripping common non-prefix suffixes such as /P, /M, /QRP, /B, and any characters following a hyphen (-).

Step 2: Exact Match (`_check_exact_match`)

The highest-priority lookup is for an exact match. The `CTY.DAT` file can contain entries prefixed with = that map a full, unique callsign to a specific entity. The script checks for these first.

Step 3: Hardcoded Special Cases (`_check_special_cases`)

The script then checks for hardcoded exceptions that do not follow standard patterns. The primary rules are:

- Any callsign ending in /MM (Maritime Mobile) is immediately classified as an "Unknown" entity.
- A specific rule identifies callsigns matching the pattern KG4[A-Z]{2} (e.g., KG4AA) as Guantanamo Bay. This prevents other, non-Guantanamo Bay KG4 callsigns from being incorrectly identified.

- A related rule invalidates any portable KG4 callsign (e.g., KG4XX/P) by classifying it as "Unknown" to prevent ambiguity.

Step 4: Portable Call Logic (`_handle_portable_call`)

If the cleaned callsign contains a /, it is processed by a dedicated handler that uses a series of heuristics to identify the `portableid`. See Section 4 for details.

Step 5: Longest Prefix Match (`_find_longest_prefix`)

If the call is not resolved by any of the previous steps, this default lookup method is used. It takes the callsign string (e.g., VP2VMM) and checks if it is a known prefix. If not, it removes the last character and tries again (VP2VM), repeating this process until it finds the longest possible valid prefix (VP2V) that exists in the CTY.DAT data. As an additional safeguard, this step includes a check to ensure that a non-Guantanamo Bay callsign (e.g., KG4ABC) does not incorrectly match the generic KG4 prefix.

4. Portable Call Heuristics

The `_handle_portable_call` method uses the following ordered checks. If a rule is satisfied, a result is returned and the process stops.

1. **Invalid digit/call Format:** The script first checks for the invalid digit/callsign format (e.g., 7/KD4D). If this pattern is found, the call is considered invalid and returns "Unknown".
2. **Unambiguous Prefix Rule:** The script checks if exactly one side of the / is a valid prefix in cty.dat while the other is not. If so, the valid side is identified as the `portableid`.
3. **"Strip the Digit" Heuristic:** If the call is still ambiguous, this tie-breaker temporarily strips a trailing digit from each side. If this makes one side a valid prefix while the other remains invalid, the original, unmodified side that produced the match is chosen as the `portableid`. This is critical for calls like HC8N/4.
4. **US/Canada Heuristic:** This rule handles the callsign/digit format for domestic US/Canada calls. If one side appears to have the structure of a US or Canadian callsign and the other is a single digit, the script identifies the **single digit** as the `portableid`.
5. **"Ends in a Digit" Heuristic:** This is the final tie-breaker. If exactly one side of the / ends in a digit while the other does not, the side ending in the digit is identified as the `portableid`. This correctly resolves calls like WT7/0L5Y.
6. **Final Action (Fall-through):** If a portable call remains ambiguous after all of the above heuristics, the `_handle_portable_call` function

concludes without a result. The main algorithm then **falls through** to the **Longest Prefix Match (Step 5)** to make a final attempt at resolving the callsign before it is classified as "Unknown".