# Documentation: GPX-to-JSON Normalizer

Auto-generated

July 30, 2025

## 1 Overview

This tool reads a GPX file and converts each track point into a compact JSON object. It keeps only the essential fields (latitude, longitude, elevation, time) and captures any extension tags (like power, heart rate, cadence) under the `<extensions>` element.

## 2 Environment Setup

- Python 3.x
- Install dependencies:

```
pip install gpxpy
```

## 3 Usage

Run the script from the command line:

```
python gpx_to_json.py <input.gpx> <output.json>
```

Where:

- `input.gpx` is the source GPX file.
- `output.json` is the JSON file to create.

## 4 Code Reference

Below is the full script with explanations.

## 4.1 Imports and Setup

```
import argparse
import os
import json
import gpxpy
```

These modules provide:

- `argparse` for parsing command-line options.
- `os` to create directories if needed.
- `json` to write the output file.
- `gpxpy` to read GPX files.

## 4.2 Argument Parsing

```
def parse_args():
    parser = argparse.ArgumentParser(
        description="Normalize a GPX file to a compact JSON
            list of track-point objects"
    )
    parser.add_argument("gpx_file", help="Path to the GPX file
        to read")
    parser.add_argument("output_json", help="Path to write the
        normalized JSON output")
    return parser.parse_args()
```

This function sets up two required arguments: the input GPX file and the output JSON path.

## 4.3 Loading GPX Data

```
def load_gpx(path):
    with open(path, 'r') as f:
        return gpxpy.parse(f)
```

Opens the given file path and uses `gpxpy` to parse it into a GPX object.

## 4.4 Converting Points

```
def gpx_to_points(gpx):
    points = []
    for track in gpx.tracks:
        for segment in track.segments:
            for pt in segment.points:
```

```
                # Each point must have at least latitude and
                    longitude
                point = {
                    "lat": pt.latitude,
                    "long": pt.longitude,
                    "elv": pt.elevation if pt.elevation is not
                        None else None,
                    "time": pt.time.isoformat() if pt.time else
                         None,
                }

                # include extensions
                if pt.extensions:
                    for ext in pt.extensions:
                        # If an extension does have children,
                            parse them individually, otherwise
                            add the key:value pair directly
                        if any(True for _ in ext):
                            for child in ext:
                                point[child.tag.split("}")[-1]]
                                     = child.text
                        else:
                            tag = ext.tag.split("}")[-1]
                            text = (
                                ext.text.strip()
                                if ext.text and ext.text.strip
                                    ()
                                else None
                            )
                            if text:
                                point[tag] = text

                # Add parsed point to list
                points.append(point)

    return points
```

This function:

1. Iterates each track, segment, and point.
2. Builds a dict with rounded coordinates, elevation, and ISO time.
3. Reads `<extensions>`: if there are nested tags, it parses each child; otherwise, it takes the direct text.
4. Appends each point dict to a list.

## 4.5 Writing JSON Output

```
def write_json(points, path):
    os.makedirs(os.path.dirname(path) or '.', exist_ok=True)
    with open(path, 'w') as f:
        json.dump(points, f, separators=(',', ':'),
            ensure_ascii=False)
```

Creates any needed folders and writes a compact JSON with no extra spaces.

## 4.6 Main Entry Point

```
def main():
    args = parse_args()
    gpx = load_gpx(args.gpx_file)
    points = gpx_to_points(gpx)
    write_json(points, args.output_json)

if __name__ == '__main__':
    main()
```

This ties everything together: parse arguments, load the GPX, convert points, and write the JSON.