Configuration File (config.py) – Tutorial & Reference

Purpose

The `config.py` file manages all configuration settings for the Supervisor (UDPServerManager) application. It handles loading, saving, and validating user preferences, device profiles, and persistent settings. This file ensures that the application can remember user choices and device information between runs.

What is a Configuration File?

A configuration file is a place where you store settings that control how the software behaves. Instead of hard-coding values in the program, you keep them in a separate file so they can be changed without editing the code.

Typical Contents

- **Network settings** (e.g., default ports, timeouts)
- **Device profiles** (e.g., names, IP addresses, roles)
- **User preferences** (e.g., window size, theme)
- **Paths** to log files, images, or other resources

Example (Python, but easy to read)

```python
# config.py
import json

CONFIG_FILE = 'settings.json'

def load_config():
    with open(CONFIG_FILE, 'r') as f:
        return json.load(f)

def save_config(config):
    with open(CONFIG_FILE, 'w') as f:
        json.dump(config, f, indent=2)
```

How Embedded C Users Can Relate

- Think of `config.py` as the equivalent of a header file or EEPROM/flash settings in embedded systems.
- Instead of #define or const variables, you use a JSON or INI file for settings.
- The Python code reads/writes these settings at startup and shutdown, just like embedded code might load/save from non-volatile memory.

Editing Configuration

- You can edit the config file with any text editor (Notepad, VS Code, etc.).
- The format is usually JSON (easy to read: key-value pairs, like a dictionary).
- Example JSON:

```
{
  "window_size": [1200, 800],
  "theme": "dark",
  "devices": [
    {"name": "TransferTapeBrake", "ip": "192.168.1.10", "role": "brake"},
    {"name": "SpeedSensor", "ip": "192.168.1.11", "role": "sensor"}
  ]
}
```

Device/Command Dictionary Management

- Each device class has its own config file in `core/workers/<device>/` and command dictionary in `shared_dictionaries/command_dictionaries/`.
- The config file defines device-specific settings; the dictionary defines commands and parameters.
- Keep these files up to date and document any changes.
- See `message_creator_panel.md` for how the UI uses these files.

Best Practices

- Always back up your config file before making changes.
- Validate the file after editing (Python will show an error if the format is wrong).
- Use comments in your documentation, but not in JSON files (JSON does not support comments).
- Use descriptive names and unique keys for all devices and parameters.
- Back up config and dictionary files before making changes.
- Test changes in the UI to ensure correct behavior.

Summary

- `config.py` is the bridge between user preferences and the running application.
- It is easy to edit and understand, even for non-Python users.
- Treat it like a settings file in embedded systems: change values to adjust behavior without touching the main code.