Franklin Battery Automation - Complete Scripts

All Scripts with Personal Information Sanitized

IMPORTANT: Replace all placeholder values with your actual credentials and IDs before using.

Table of Contents

Core Automation Scripts

daily*status*report.py

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Daily Battery Status Report
Sends summary of the day's solar intelligence decisions and current status
"""
import subprocess
from datetime import datetime, timedelta

def get_battery_status():
    """Get current battery status"""
    try:
        result = subprocess.run(
            ['/volume1/docker/franklin/get_battery_status.py'],
            capture_output=True,
            text=True,
            timeout=30
        )
        return result.stdout
    except Exception as e:
        return f"ERROR getting battery status: {e}"

def get_todays_intelligence_log():
    """Get today's solar intelligence decisions"""
    today = datetime.now().strftime("%Y-%m-%d")
    try:
        with open('/volume1/docker/franklin/logs/solar_intelligence.log', 'r') as f:
            lines = f.readlines()

        todays_lines = [line for line in lines if line.startswith(today)]

        if todays_lines:
            return ''.join(todays_lines)
        else:
            return "No solar intelligence activity logged today."

    except Exception as e:
        return f"ERROR reading intelligence log: {e}"

def get_todays_energy_summary():
    """Get today's energy flow summary from continuous monitoring"""
    today = datetime.now().strftime("%Y-%m-%d")
    try:
        result = subprocess.run(
            ['grep', today, '/volume1/docker/franklin/logs/continuous_monitoring.csv'],
            capture_output=True,
            text=True
        )

        lines = result.stdout.strip().split('\n')
        if not lines or lines[0] == '':
            return "No monitoring data available for today."

        # Parse data to get summary
        soc_values = []
        solar_values = []
        grid_values = []
        battery_values = []

        for line in lines:
            if not line or 'timestamp' in line:
                continue
            parts = line.split(',')
            if len(parts) >= 5:
                try:
                    soc_values.append(float(parts[1]))
                    solar_values.append(float(parts[2]))
                    grid_values.append(float(parts[3]))
                    battery_values.append(float(parts[4]))
                except:
                    continue

        if soc_values:
            summary = f"""
Today's Energy Summary (based on {len(soc_values)} readings):
  SOC Range: {min(soc_values):.1f}% - {max(soc_values):.1f}%
  Current SOC: {soc_values[-1]:.1f}%
  Solar Production:
    Average: {sum(solar_values)/len(solar_values):.2f} kW
    Peak: {max(solar_values):.2f} kW

  Grid Usage:
    Average: {sum(grid_values)/len(grid_values):.2f} kW
    Peak Import: {max(grid_values):.2f} kW

  Battery Activity:
    Peak Charge: {min(battery_values):.2f} kW
    Peak Discharge: {max(battery_values):.2f} kW
"""
            return summary
        else:
            return "No valid monitoring data for today."

    except Exception as e:
        return f"ERROR analyzing energy data: {e}"

def main():
    print("="*70)
    print("FRANKLIN BATTERY - DAILY STATUS REPORT")
    print(f"Generated: {datetime.now().strftime('%Y-%m-%d %I:%M %p')}")
    print("="*70)
    print()

    # Current battery status
    print("CURRENT BATTERY STATUS:")
    print("-"*70)
    print(get_battery_status())
    print()

    # Today's energy summary
    print("TODAY'S ENERGY SUMMARY:")
    print("-"*70)
    print(get_todays_energy_summary())
    print()

    # Today's solar intelligence decisions
    print("TODAY'S SOLAR INTELLIGENCE DECISIONS:")
    print("-"*70)
    print(get_todays_intelligence_log())
    print()

    # Peak period readiness
    now = datetime.now()
    peak_start = now.replace(hour=17, minute=0, second=0, microsecond=0)

    if now < peak_start:
        time_to_peak = peak_start - now
        hours = int(time_to_peak.total_seconds() / 3600)
        minutes = int((time_to_peak.total_seconds() % 3600) / 60)
        print(f"PEAK PERIOD STATUS:")
        print("-"*70)
        print(f"Peak period starts in: {hours}h {minutes}m (5:00 PM)")
        print()
    else:
        print(f"PEAK PERIOD STATUS:")
        print("-"*70)
        print("Currently in peak period (5:00 PM - 8:00 PM)")
        print()

    print("="*70)
    print("End of Report")
    print("="*70)

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

Data Collection Scripts

---

collect_pvoutput.py

⚠️ **REPLACE THESE VALUES:** - `API_KEY` : Your PVOutput API key - `GROUND_MOUNT_SID` : Your first system ID (or remove if you only have one) - `HOUSE_SID` : Your second system ID

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
PVOutput Daily Data Collector
Collects yesterday's completed solar production data
"""
import requests
import csv
from datetime import datetime, timedelta
from pathlib import Path

# ⚠️ REPLACE WITH YOUR PVOUTPUT CREDENTIALS
API_KEY = "YOUR_PVOUTPUT_API_KEY_HERE"
GROUND_MOUNT_SID = "YOUR_SYSTEM_ID_1"  # Remove if you only have one system
HOUSE_SID = "YOUR_SYSTEM_ID_2"

LOG_DIR = Path("/volume1/docker/franklin/logs")
GROUND_LOG = LOG_DIR / "pvoutput_ground_mount_daily.csv"
HOUSE_LOG = LOG_DIR / "pvoutput_house_daily.csv"

def get_and_save_daily_output(system_id, filepath, system_name, date):
    """Get daily output and save to CSV"""
    url = "https://pvoutput.org/service/r2/getoutput.jsp"
    headers = {
        "X-Pvoutput-Apikey": API_KEY,
        "X-Pvoutput-SystemId": system_id
    }
    params = {
        "d": date.strftime("%Y%m%d")
    }

    try:
        response = requests.get(url, headers=headers, params=params, timeout=30)
        if response.status_code != 200:
            print(f"Error for {system_name}: HTTP {response.status_code}")
            return False

        # Split by SEMICOLONS first (multiple records), then by commas (fields within record)
        records = response.text.strip().split(';')

        for record in records:
            # Split each record by commas to get individual fields
            parts = record.split(',')

            if len(parts) < 6:
                continue

            # Extract only the fields we need (first 8)
            date_str = parts[0]

            try:
                energy = float(parts[1]) if parts[1] and parts[1] != 'NaN' else 0.0
                efficiency = float(parts[2]) if parts[2] and parts[2] != 'NaN' else 0.0
                exported = float(parts[3]) if parts[3] and parts[3] != 'NaN' else 0.0
                used = float(parts[4]) if parts[4] and parts[4] != 'NaN' else 0.0
                peak_power = float(parts[5]) if parts[5] and parts[5] != 'NaN' else 0.0
            except ValueError:
                continue

            peak_time = parts[6] if len(parts) > 6 else ''
            condition = parts[7] if len(parts) > 7 else ''

            # Check if already exists
            if filepath.exists():
                with open(filepath, 'r') as f:
                    if date_str in f.read():
                        continue

            # Append to CSV
            with open(filepath, 'a', newline='') as f:
                writer = csv.writer(f)
                writer.writerow([date_str, energy, efficiency, exported, used, peak_power, peak_time, condition])

            print(f"✓ Saved {date_str} to {system_name} ({energy} Wh)")

        return True

    except Exception as e:
        print(f"Error for {system_name}: {e}")
        return False

def main():
    yesterday = datetime.now() - timedelta(days=1)
    print(f"Collecting PVOutput data for {yesterday.strftime('%Y-%m-%d')}")

    get_and_save_daily_output(GROUND_MOUNT_SID, GROUND_LOG, "Ground Mount", yesterday)
    get_and_save_daily_output(HOUSE_SID, HOUSE_LOG, "House", yesterday)

    print("✓ Complete")

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

collect_weather.py

⚠️ **REPLACE THESE VALUES:** - `PWS_ID`: Your Weather Underground Personal Weather Station ID - `API_KEY`: Your Weather Underground API key

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Weather Underground Data Collector
Pulls weather data from your PWS and stores to CSV
"""
import requests
import csv
from datetime import datetime
from pathlib import Path

# ⚠️ REPLACE WITH YOUR WEATHER UNDERGROUND CREDENTIALS
PWS_ID = "YOUR_WEATHER_STATION_ID"
API_KEY = "YOUR_WEATHER_UNDERGROUND_API_KEY"

# File path
LOG_DIR = Path("/volume1/docker/franklin/logs")
WEATHER_LOG = LOG_DIR / "weather_data.csv"

def get_current_conditions():
    """Get current weather conditions from Weather Underground"""
    url = "https://api.weather.com/v2/pws/observations/current"

    params = {
        "stationId": PWS_ID,
        "format": "json",
        "units": "e",  # English units
        "apiKey": API_KEY
    }

    try:
        response = requests.get(url, params=params, timeout=10)
        response.raise_for_status()

        data = response.json()

        if "observations" in data and len(data["observations"]) > 0:
            obs = data["observations"][0]
            imperial = obs.get("imperial", {})

            weather_data = {
                'timestamp': datetime.now().isoformat(),
                'obs_time_local': obs.get('obsTimeLocal', ''),
                'station_id': obs.get('stationID', PWS_ID),
                'neighborhood': obs.get('neighborhood', ''),
                'temp_f': imperial.get('temp'),
                'heat_index_f': imperial.get('heatIndex'),
                'dewpoint_f': imperial.get('dewpt'),
                'wind_chill_f': imperial.get('windChill'),
                'humidity': obs.get('humidity'),
                'pressure_inhg': imperial.get('pressure'),
                'wind_speed_mph': imperial.get('windSpeed'),
                'wind_gust_mph': imperial.get('windGust'),
                'wind_dir_degrees': obs.get('winddir'),
                'precip_rate_in_hr': imperial.get('precipRate'),
                'precip_total_in': imperial.get('precipTotal'),
                'solar_radiation_wm2': obs.get('solarRadiation'),
                'uv_index': obs.get('uv'),
            }

            return weather_data
        else:
            print(f"No observation data available")
            return None

    except requests.exceptions.RequestException as e:
        print(f"Error fetching weather data: {e}")
        return None
    except Exception as e:
        print(f"Error parsing weather data: {e}")
        return None

def save_to_csv(weather_data):
    """Save weather data to CSV file"""
    if not weather_data:
        return

    WEATHER_LOG.parent.mkdir(parents=True, exist_ok=True)
    file_exists = WEATHER_LOG.exists()

    fieldnames = [
        'timestamp', 'obs_time_local', 'station_id', 'neighborhood',
        'temp_f', 'heat_index_f', 'dewpoint_f', 'wind_chill_f',
        'humidity', 'pressure_inhg',
        'wind_speed_mph', 'wind_gust_mph', 'wind_dir_degrees',
        'precip_rate_in_hr', 'precip_total_in',
        'solar_radiation_wm2', 'uv_index'
    ]

    try:
        with open(WEATHER_LOG, 'a', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=fieldnames)
            if not file_exists:
                writer.writeheader()
            writer.writerow(weather_data)

        print(f"✓ Weather data saved: {weather_data['temp_f']}°F, {weather_data['solar_radiation_wm2']} W/m² solar")

    except Exception as e:
        print(f"Error saving weather data: {e}")

def collect_weather():
    """Main collection function"""
    weather_data = get_current_conditions()
    if weather_data:
        save_to_csv(weather_data)
        return True
    return False

if __name__ == "__main__":
    import sys
    success = collect_weather()
    sys.exit(0 if success else 1)
```

continuous_monitor.py

⚠️ **REPLACE THESE VALUES:** - `USERNAME` : Your Franklin WH account email - `PASSWORD` : Your Franklin WH account password - `GATEWAY_ID` : Your Franklin gateway ID (from mobile app)

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Continuous Battery Monitoring
Logs battery stats every 15 minutes indefinitely
"""
import asyncio
import csv
from datetime import datetime
from franklinwh import Client, TokenFetcher

# ⚠️ REPLACE WITH YOUR FRANKLIN WH CREDENTIALS
USERNAME = "YOUR_EMAIL@example.com"
PASSWORD = "YOUR_PASSWORD"
GATEWAY_ID = "YOUR_GATEWAY_ID"

LOG_FILE = "/volume1/docker/franklin/logs/continuous_monitoring.csv"

async def continuous_monitor(interval_minutes=15):
    """
    Continuously monitor battery stats and log to CSV.
    Runs indefinitely until stopped.

    Args:
        interval_minutes: How often to log data (default 15 minutes)
    """
    fetcher = TokenFetcher(USERNAME, PASSWORD)
    client = Client(fetcher, GATEWAY_ID)

    # Check if file exists
    file_exists = False
    try:
        with open(LOG_FILE, 'r') as f:
            file_exists = True
    except FileNotFoundError:
        pass

    print("=" * 70)
    print("CONTINUOUS BATTERY MONITORING")
    print("=" * 70)
    print(f"Interval: {interval_minutes} minutes")
    print(f"Log file: {LOG_FILE}")
    print(f"Started: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print("=" * 70)
    print("\nPress Ctrl+C to stop monitoring")
    print("\nLogging data...\n")

    iteration = 0
    with open(LOG_FILE, 'a', newline='') as csvfile:
        fieldnames = ['timestamp', 'soc_percent', 'solar_kw', 'grid_kw',
                      'battery_kw', 'home_load_kw', 'grid_status',
                      'battery_charge_total', 'battery_discharge_total',
                      'grid_import_total', 'solar_total']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        if not file_exists:
            writer.writeheader()
            csvfile.flush()

        try:
            while True:
                iteration += 1
                try:
                    stats = await client.get_stats()
                    now = datetime.now()

                    data = {
                        'timestamp': now.strftime('%Y-%m-%d %H:%M:%S'),
                        'soc_percent': f'{stats.current.battery_soc:.2f}',
                        'solar_kw': f'{stats.current.solar_production:.3f}',
                        'grid_kw': f'{stats.current.grid_use:.3f}',
                        'battery_kw': f'{stats.current.battery_use:.3f}',
                        'home_load_kw': f'{stats.current.home_load:.3f}',
                        'grid_status': stats.current.grid_status.name,
                        'battery_charge_total': f'{stats.totals.battery_charge:.3f}',
                        'battery_discharge_total': f'{stats.totals.battery_discharge:.3f}',
                        'grid_import_total': f'{stats.totals.grid_import:.3f}',
                        'solar_total': f'{stats.totals.solar:.3f}'
                    }

                    writer.writerow(data)
                    csvfile.flush()

                    # Display progress
                    print(f"[{iteration:04d}] {now.strftime('%m/%d %H:%M')} | "
                          f"SOC: {stats.current.battery_soc:5.2f}% | "
                          f"Solar: {stats.current.solar_production:6.3f}kW | "
                          f"Grid: {stats.current.grid_use:6.3f}kW | "
                          f"Battery: {stats.current.battery_use:+6.3f}kW | "
                          f"Load: {stats.current.home_load:6.3f}kW")

                except Exception as e:
                    print(f"[{iteration:04d}] Error: {e}")

                # Wait for next interval
                await asyncio.sleep(interval_minutes * 60)

        except KeyboardInterrupt:
            print("\n" + "=" * 70)
            print("MONITORING STOPPED")
            print("=" * 70)
            print(f"Stopped: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
            print(f"Total iterations: {iteration}")
            print(f"Log file: {LOG_FILE}")
            print("=" * 70)
```

```
if __name__ == "__main__":
    # Log every 15 minutes by default
    asyncio.run(continuous_monitor(interval_minutes=15))
```

Utility Scripts

get*battery*status.py

⚠️ **REPLACE THESE VALUES:** - `USERNAME` : Your Franklin WH account email - `PASSWORD` : Your Franklin WH account password - `GATEWAY_ID` : Your Franklin gateway ID

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Get Current Battery Status
Quick utility to check battery SOC and power flows
"""
import asyncio
from franklinwh import Client, TokenFetcher

# ⚠️ REPLACE WITH YOUR FRANKLIN WH CREDENTIALS
USERNAME = "YOUR_EMAIL@example.com"
PASSWORD = "YOUR_PASSWORD"
GATEWAY_ID = "YOUR_GATEWAY_ID"

async def main():
    try:
        fetcher = TokenFetcher(USERNAME, PASSWORD)
        client = Client(fetcher, GATEWAY_ID)

        stats = await client.get_stats()

        print("=" * 50)
        print("FRANKLIN BATTERY STATUS")
        print("=" * 50)
        print(f"Battery SOC:        {stats.current.battery_soc:.1f}%")
        print(f"Solar Production:   {stats.current.solar_production:.3f} kW")
        print(f"Grid Use:           {stats.current.grid_use:.3f} kW")
        print(f"Battery Use:        {stats.current.battery_use:.3f} kW")
        print(f"Home Load:          {stats.current.home_load:.3f} kW")
        print(f"Grid Status:        {stats.current.grid_status.name}")
        print("=" * 50)

        # Return SOC for use in automation
        return stats.current.battery_soc

    except Exception as e:
        print(f"Error: {e}")
        return None
```

```
if __name__ == "__main__":
    soc = asyncio.run(main())
```

switch*to*backup_v2.py

⚠️ **REPLACE THESE VALUES:** - `USERNAME` : Your Franklin WH account email - `PASSWORD` : Your Franklin WH account password - `GATEWAY_ID` : Your Franklin gateway ID

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Switch to Emergency Backup Mode
This starts grid charging the battery
"""
import asyncio
from franklinwh import Client, TokenFetcher, Mode

# ⚠️ REPLACE WITH YOUR FRANKLIN WH CREDENTIALS
USERNAME = "YOUR_EMAIL@example.com"
PASSWORD = "YOUR_PASSWORD"
GATEWAY_ID = "YOUR_GATEWAY_ID"

async def main():
    try:
        print("Authenticating with Franklin WH...")
        fetcher = TokenFetcher(USERNAME, PASSWORD)

        print("Creating client...")
        client = Client(fetcher, GATEWAY_ID)

        print("Switching to Emergency Backup mode...")
        await client.set_mode(Mode.emergency_backup())

        print("✓ Successfully switched to Emergency Backup mode")
        print("✓ Battery is now charging from grid")
    except Exception as e:
        print(f"✗ Error: {e}")
        import traceback
        traceback.print_exc()

if __name__ == "__main__":
    asyncio.run(main())
```

switch*to*tou_v2.py

⚠️ **REPLACE THESE VALUES:** - `USERNAME` : Your Franklin WH account email - `PASSWORD` : Your Franklin WH account password - `GATEWAY_ID` : Your Franklin gateway ID

```
#!/volume1/docker/franklin/venv311/bin/python3
"""
Switch to Time-of-Use Mode
This stops grid charging and returns to TOU operation
```

```
"""
import asyncio
from franklinwh import Client, TokenFetcher, Mode

# ⚠️ REPLACE WITH YOUR FRANKLIN WH CREDENTIALS
USERNAME = "YOUR_EMAIL@example.com"
PASSWORD = "YOUR_PASSWORD"
GATEWAY_ID = "YOUR_GATEWAY_ID"

async def main():
    try:
        print("Authenticating with Franklin WH...")
        fetcher = TokenFetcher(USERNAME, PASSWORD)

        print("Creating client...")
        client = Client(fetcher, GATEWAY_ID)

        print("Switching to TOU mode...")
        await client.set_mode(Mode.time_of_use())

        print("✓ Successfully switched to TOU mode")
        print("✓ Battery charging stopped, using TOU schedule")
    except Exception as e:
        print(f"✗ Error: {e}")
        import traceback
        traceback.print_exc()

if __name__ == "__main__":
    asyncio.run(main())
```

Configuration Files

requirements.txt

```
franklinwh==0.13.0
requests>=2.31.0
```

Installation Commands

```
# Create virtual environment
python3 -m venv venv311

# Activate it
source venv311/bin/activate

# Install dependencies
pip install --break-system-packages -r requirements.txt
```

Quick Reference

File Locations

```
/volume1/docker/franklin/
├── venv311/                          # Python virtual environment
├── logs/                             # All log files
│   ├── solar_intelligence.log
│   ├── continuous_monitoring.csv
│   ├── weather_data.csv
│   ├── pvoutput_house_daily.csv
│   └── pvoutput_ground_mount_daily.csv
├── requirements.txt
├── morning_solar_intelligence.py
├── midday_charge_check.py
├── final_safety_check.py
├── daily_status_report.py
├── collect_pvoutput.py
├── collect_weather.py
├── continuous_monitor.py
├── get_battery_status.py
├── switch_to_backup_v2.py
└── switch_to_tou_v2.py
```

Credentials Summary

You need to replace:

1. **Franklin WH** (in 5 scripts):
    - USERNAME
    - PASSWORD
    - GATEWAY_ID
2. **PVOutput** (in 1 script):
    - API_KEY
    - System IDs (GROUND*MOUNT*SID, HOUSE_SID)
3. **Weather Underground** (in 1 script):
    - PWS_ID
    - API_KEY

Testing Checklist

After configuring all scripts:

```
cd /volume1/docker/franklin
source venv311/bin/activate

# Test each script
./get_battery_status.py
./collect_weather.py
./collect_pvoutput.py
./morning_solar_intelligence.py
./midday_charge_check.py
./final_safety_check.py
./daily_status_report.py

# Test mode switching (⚠️ will actually change modes!)
./switch_to_backup_v2.py
./switch_to_tou_v2.py

# Start continuous monitoring (Ctrl+C to stop)
./continuous_monitor.py
```

Support & Community

Getting Help

1. Check logs in `/volume1/docker/franklin/logs/`
2. Test scripts manually
3. Verify credentials are correct
4. Check Task Scheduler history

Sharing Your Success

If this system works for you: - Document your savings - Share your customizations - Help others in the community - Report bugs or improvements

---

Last Updated: January 2, 2026
Version: 1.0
Author: Ken Pauley

END OF SCRIPTS