HTML Content Scraper Development Guide

IODA Team

July 16, 2025

Contents

1 Core Architecture

1.1 Class Structure

```python
class ProviderName:
    def __init__(self):
        # URLs, dates, paths

    def check_folder(self, type):
        # Directory creation (keep this name - it's used everywhere)

    def scrape(self):
        # Main entry point - does everything
```

1.2 Before You Start - Information to Collect

1. **Target URL** – The exact webpage containing outage data
2. **Data Location** – How data is structured (tables, divs, lists)
3. **Update Schedule** – How often the site updates (daily, weekly, etc.)
4. **Data Fields** – What information is available (dates, times, areas, etc.)
5. **Site Quirks** – Any special requirements (SSL issues, headers, etc.)

1.3 Critical Architectural Decisions

1.3.1 Processing Strategy

Always save raw HTML files AND process immediately. Both steps are required.

```python
def scrape(self):
    # Fetch data
    response = requests.get(self.url)

    # Save raw file - ALWAYS KEEP THIS
    with open(file_path, "w", encoding="utf-8") as file:
        file.write(response.text)

    # Process immediately - ALWAYS DO THIS TOO
    processor = Process_Provider(self.year, self.month, self.today, file_path)
    processor.run()
```

1.3.2 Error Handling Strategy

```python
# Use this pattern consistently
response = requests.get(self.url)
if response.status_code == 200:
    # Process
else:
    print(f"Failed to retrieve webpage. Status code: {response.status_code}")
    return  # Don't crash, just exit gracefully
```

1.3.3 Directory Structure

Non-negotiable pattern:

`{country}/{provider}/raw/{year}/{month}/`
`{country}/{provider}/processed/{year}/{month}/`

Implementation:

```
def check_folder(self, type):
    self.folder_path = f"./{country}/{provider}/" + type + "/" + self.year + "/" + self.month
    os.makedirs(self.folder_path, exist_ok=True)
```

Keep this exact signature - it's used across all scrapers.

2 HTML Parsing Patterns

2.1 Pattern 1: Table-Based Data

Used by sites with structured HTML tables:

```
def parse_table_data(self, soup):
    table = soup.find('table')  # or find_all if multiple tables
    if not table:
        return []

    rows = table.find_all('tr')
    data = []
    for row in rows[1:]:  # Skip header
        cols = row.find_all('td')
        if len(cols) >= expected_columns:
            # Extract data from columns
            data.append(extracted_item)
    return data
```

2.2 Pattern 2: Container-Based Data

Used by sites with div containers or card layouts:

```
def parse_container_data(self, soup):
    containers = soup.find_all('div', class_='outage-item')  # Adjust selector
    data = []
    for container in containers:
        # Extract from each container
        title = container.find('h3')
        if title:  # Always check if elements exist
            # Process
    return data
```

2.3 Pattern 3: Mixed Content

Used by sites with irregular structure:

```
def parse_mixed_content(self, soup):
    # Find all potential data containers
    # Use flexible selectors
    # Validate each piece of data
    # Handle missing elements gracefully
```

3 Processor Requirements

3.1 Processor Structure

Every HTML scraper will have a processor class:

```python
class Process_Provider:
    def __init__(self, year, month, today, file_path):
        # Standard signature - don't change this

    def check_folder(self, type):
        # Directory management for processed data

    def get_data(self):
        # Parse the HTML file

    def save_json(self, data):
        # Save to standard JSON format

    def run(self):
        # Orchestrate everything
```

3.2 Command-Line Processing

The process_file.py script supports batch processing:

```
python process_file.py \
    --country india \
    --provider npp \
    --start_date 2025-04-18 \
    --end_date 2025-04-22
```

For manual processor execution (mainly Ukraine):

```python
# Modify file path and folder path in processor
file = "path/to/raw/file.html"
self.folder_path = "path/to/processed/output/"
```

3.3 Standard Output Format

Format:

```
  {
    "country": "Country Name",
    "start": "YYYY-MM-DD_HH-MM-SS",
    "end": "YYYY-MM-DD_HH-MM-SS",
    "duration_(hours)": 2.5,
    "event_category": "planned maintenance",
    "area_affected": {
      "Region/State": ["Area1", "Area2", "Area3"]
    }
  }
```

3.3.1 DateTime Format Rules

- Always use ISO date format for the date part
- Always use underscore separator: `YYYY-MM-DD_HH-MM-SS`
- Use 24-hour time format
- Duration in decimal hours (2.5 = 2 hours 30 minutes)

4 Configuration Requirements

4.1 Ukraine-Specific Configuration

For Ukraine providers, modify constants.py:

```
# Set root directory for data storage
root_dir = "/path/to/your/data/directory"
year = "2025"
month = "01"
date = "01"
```

Important: You must change the "root_dir" in constants.py to where you plan to save data.

4.2 Regional Execution Differences

4.2.1 India, Nigeria, Pakistan

- Processors are automatically called when running scraper files
- Scrapers can be directly executed
- Integrated processing approach

4.2.2 Ukraine

- Scrapers and processors run separately
- Uses constants.py for configuration
- Post-processor must be executed manually after scraping
- Oblast-based directory structure

5 File Naming Standards

5.1 Actual Pattern in Use

`power_outages.{COUNTRY_CODE}.{provider}.raw.{YYYY-MM-DD}.html`
`power_outages.{COUNTRY_CODE}.{provider}.processed.{YYYY-MM-DD}.json`

5.2 Country Codes

Country codes that actually exist:

- IND (India)
- NG (Nigeria)
- PK (Pakistan)
- UA (Ukraine)
- CM (Cameroon)

6 Real-World Handling

6.1 Sites with No Data

```
# Always handle this case
if not table or len(rows) == 0:
    print(f"No outage data found for {self.today}")
    # Still save an empty processed file
    self.save_json([])
    return
```

6.2 Sites with Irregular HTML

```
# Always validate before accessing
element = soup.find('div', class_='data')
if element:
    text = element.get_text(strip=True)
    if text:  # Check if text exists and isn't empty
        # Process
```

7 Integration Requirements

7.1 Adding to daily_scraping.py

```
# Import
from {country}.{provider}.{provider} import {ProviderClass}

# In scrape() function
try:
    provider_scraper = {ProviderClass}()
    provider_scraper.scrape()
except Exception as e:
    print(f"Failed to scrape outage data from {Provider}.")
```

Note: It is recommended to run daily_scraping.py near noon time EST to avoid intermittent internet issues.

7.2 Adding to monthly_scraping.py

For providers that offer monthly outage data:

```
# Import
from {country}.{provider}.{provider} import {ProviderClass}

# In scrape() function
try:
    provider_scraper = {ProviderClass}()
    provider_scraper.scrape()
except Exception as e:
    print(f"Failed to scrape monthly data from {Provider}.")
```

7.3 Adding to process_file.py

```
# Add to provider lists
{country}_providers = ["existing1", "existing2", "{new_provider}"]

# Add processing logic
```

```
elif provider == "{new_provider}":
    process = Process_{Provider}(year, month, date, file_path)
    process.run()
```

8 Development Process

8.1 Analysis Phase

- **Manually visit the target site**
- **View source to understand HTML structure**
- **Check if data updates consistently**
- **Note any authentication/headers needed**

8.2 Implementation Order

1. Create basic scraper class with `__init__` and `scrape`
2. Test URL fetching and HTML saving
3. Create processor class and test parsing
4. Integrate both pieces
5. Test with multiple dates

8.3 Testing Requirements

- **Test with at least 3 different dates**
- **Test with a day that has no data**
- **Verify file naming matches convention**
- **Check output JSON format**

9 Success Criteria

A successful HTML content scraper:

1. **Follows naming conventions exactly**
2. **Produces standard JSON output**
3. **Handles missing data gracefully**
4. **Integrates with existing scripts**
5. **Works reliably across multiple dates**

10 Additional Resources

- **Detailed workflow:** /docs/workflow.md
- **Notion documentation:** See repository README for link
- **Repository structure:** Follows country/provider pattern
- **Data analysis:** Available in Data Analysis directory with KeepItOn raw CSV data

10.1 Repository Structure Overview

```
data_analysis/          # Raw CSV data and analysis
{country}/              # Country-specific directories
   {provider}/         # Provider-specific code
       {provider}.py   # Main scraper
       process_{provider}.py  # Processor
docs/
   workflow.md         # Detailed workflow
daily_scraping.py       # Daily automation
monthly_scraping.py     # Monthly automation
process_file.py         # Batch processing
```