# scan.py — OERForge Static Site Asset and Content Scanner

## Overview

scan.py provides the core logic for scanning site content, extracting assets, and populating the SQLite database for the OERForge project. It supports hierarchical TOC parsing, asset extraction from Markdown, Jupyter Notebooks, and DOCX files, and maintains relationships between pages, sections, and files. The module is designed for clarity, extensibility, and ease of debugging, with standardized logging and comprehensive docstrings.

## Module Docstring

"""  
scan.py  
--------  
Static Site Asset and Content Scanner  
  
This module provides functions for scanning site content, extracting assets, and populating the SQLite database  
with page, section, and file records. It supports Markdown, Jupyter Notebooks, and DOCX files, and maintains  
hierarchical relationships from the Table of Contents (TOC) YAML. All functions are documented and organized for  
clarity and maintainability. Logging is standardized and all major operations are traced for debugging.  
  
Key Features:  
- Batch reading and asset extraction for supported file types  
- Hierarchical TOC walking and database population  
- Asset linking and MIME type detection  
- Section and descendant queries using recursive CTEs  
  
Usage:  
 Import and call scan\_toc\_and\_populate\_db(config\_path) to scan the TOC and populate the database.  
 Use get\_descendants\_for\_parent() to query section hierarchies.  
"""

## Functions

### log\_event(message, level="INFO")

def log\_event(message, level="INFO"):  
 """  
 Logs an event to both stdout and scan.log in the project root.  
 Uses Python logging and project-standard setup.  
 """

**Purpose:**  
Standardizes logging for all scan operations, ensuring traceability in both the console and log files.

### batch\_read\_files(file\_paths)

def batch\_read\_files(file\_paths):  
 """  
 Reads multiple files and returns their contents as a dict: {path: content}  
 Supports markdown (.md), notebook (.ipynb), docx (.docx), and other file types.  
 """

**Purpose:**  
Efficiently loads content from a list of files, supporting multiple formats for downstream asset extraction.

### read\_markdown\_file(path)

def read\_markdown\_file(path):  
 """  
 Reads a markdown (.md) file and returns its content as a string.  
 """

**Purpose:**  
Loads Markdown content for asset extraction and database population.

### read\_notebook\_file(path)

def read\_notebook\_file(path):  
 """  
 Reads a Jupyter notebook (.ipynb) file and returns its content as a dict.  
 """

**Purpose:**  
Loads Jupyter notebook content for asset extraction and database population.

### read\_docx\_file(path)

def read\_docx\_file(path):  
 """  
 Reads a docx file and returns its text content as a string.  
 Requires python-docx to be installed.  
 """

**Purpose:**  
Loads DOCX content for asset extraction and database population.

### batch\_extract\_assets(contents\_dict, content\_type, \*\*kwargs)

def batch\_extract\_assets(contents\_dict, content\_type, \*\*kwargs):  
 """  
 Extracts assets from multiple file contents in one pass.  
 contents\_dict: {path: content}  
 content\_type: 'markdown', 'notebook', 'docx', etc.  
 Returns a dict: {path: [asset\_records]}  
 """

**Purpose:**  
Extracts and records all linked assets (images, files, etc.) from a batch of content files, updating the database and linking files to pages.

### extract\_linked\_files\_from\_markdown\_content(md\_text, page\_id=None)

def extract\_linked\_files\_from\_markdown\_content(md\_text, page\_id=None):  
 """  
 Extract asset links from markdown text.  
 Args:  
 md\_text (str): Markdown content.  
 page\_id (optional): Page identifier for DB linking.  
 Returns:  
 list: File record dicts for each asset found.  
 """

**Purpose:**  
Finds and returns all asset links in Markdown content for database tracking.

### extract\_linked\_files\_from\_notebook\_cell\_content(cell, nb\_path=None)

def extract\_linked\_files\_from\_notebook\_cell\_content(cell, nb\_path=None):  
 """  
 Extract asset links from a notebook cell.  
 Args:  
 cell (dict): Notebook cell.  
 nb\_path (str, optional): Notebook file path.  
 Returns:  
 list: File record dicts for each asset found.  
 """

**Purpose:**  
Finds and returns all asset links in notebook cells, including embedded images and code-generated outputs.

### extract\_linked\_files\_from\_docx\_content(docx\_path, page\_id=None)

def extract\_linked\_files\_from\_docx\_content(docx\_path, page\_id=None):  
 """  
 Extract asset links from a DOCX file.  
 Args:  
 docx\_path (str): Path to DOCX file.  
 page\_id (optional): Page identifier for DB linking.  
 Returns:  
 list: File record dicts for each asset found.  
 """

**Purpose:**  
Finds and returns all asset links and embedded images in DOCX files for database tracking.

### populate\_site\_info\_from\_config(config\_filename='\_config.yml')

def populate\_site\_info\_from\_config(config\_filename='\_config.yml'):  
 """  
 Populate the site\_info table from the given config file (default: \_config.yml).  
 Args:  
 config\_filename (str): Name of the config file (e.g., '\_config.yml').  
 """

**Purpose:**  
Reads site metadata from the config YAML and updates the site\_info table in the database.

### get\_possible\_conversions(extension)

def get\_possible\_conversions(extension):  
 """  
 Get possible conversion flags for a given file extension.  
 Args:  
 extension (str): File extension (e.g., '.md', '.ipynb').  
 Returns:  
 dict: Conversion capability flags.  
 """

**Purpose:**  
Determines which conversion operations are supported for a given file type.

### scan\_toc\_and\_populate\_db(config\_path)

def scan\_toc\_and\_populate\_db(config\_path):  
 """  
 Walk the TOC from the config YAML, read each file, extract assets/images, and populate the DB with content and asset records.  
 Maintains TOC hierarchy and section relationships.  
 Args:  
 config\_path (str): Path to the config YAML file.  
 """

**Purpose:**  
The main entry point for scanning the TOC, reading files, extracting assets, and populating the database with hierarchical content and file records.

### get\_descendants\_for\_parent(parent\_output\_path, db\_path)

def get\_descendants\_for\_parent(parent\_output\_path, db\_path):  
 """  
 Query all children, grandchildren, and deeper descendants for a given parent\_output\_path using a recursive CTE.  
 Args:  
 parent\_output\_path (str): Output path of the parent section.  
 db\_path (str): Path to the SQLite database.  
 Returns:  
 list: Dicts for each descendant (id, title, output\_path, parent\_output\_path, slug, level).  
 """

**Purpose:**  
Efficiently queries the database for all descendants of a section, supporting hierarchical navigation and index generation.

## Typical Workflow

1. **Scan and Populate Database**
   * Call scan\_toc\_and\_populate\_db(config\_path) to walk the TOC, read files, extract assets, and populate the database.
2. **Asset Extraction**
   * Use batch\_extract\_assets and related helpers to extract and link assets from Markdown, Notebooks, and DOCX files.
3. **Site Metadata**
   * Use populate\_site\_info\_from\_config to update site-wide metadata in the database.
4. **Section Hierarchy Queries**
   * Use get\_descendants\_for\_parent to retrieve all children and grandchildren for a given section, supporting navigation and index pages.

## Logging and Debugging

All major operations are logged using log\_event, with output to both stdout and scan.log. This ensures traceability for debugging and auditing. Errors, warnings, and key events are clearly marked.

## Extensibility

* New asset types and file formats can be added by extending the asset extraction helpers.
* The TOC walking logic supports arbitrary nesting and slug overrides.
* All functions are documented and organized for easy onboarding and contribution.

## Intended Audience

* Contributors to OERForge
* Developers building or extending static site generators
* Anyone needing robust asset and content scanning for static sites