`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
with page, section, and file records. It supports Markdown, Jupyter Notebooks, and DOCX fil
hierarchical relationships from the Table of Contents (TOC) YAML. All functions are documen
clarity and maintainability. Logging is standardized and all major operations are traced fo

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
    Use get_descendants_for_parent() to query section hierarchies.
"""
```

Functions

`log_event(message, level="INFO")`

```python
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)`

```python
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)`

```python
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)`

```python
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)`

```python
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)`

```python
def scan_toc_and_populate_db(config_path):
    """
    Walk the TOC from the config YAML, read each file, extract assets/images, and populate
    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)`

```python
def get_descendants_for_parent(parent_output_path, db_path):
    """
    Query all children, grandchildren, and deeper descendants for a given parent_output_pat
    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,
    """
```

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

---