BrickLink Scraper Documentation

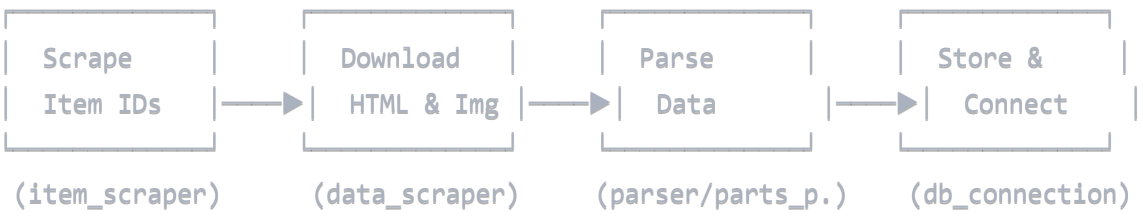
Project Overview

This project is a comprehensive data scraper for Bricklink.com, designed to collect, parse, and store information about LEGO sets, minifigures, and parts in a structured SQLite database. The system follows a multi-stage approach to gather and process data with special attention to performance, data integrity, and respectful web scraping practices.

The primary goal is to create a complete database of LEGO products that can be used for various purposes including inventory management, price analysis, and set composition studies. The data is organized in a 3rd normal form database structure with proper relationships between entities.

System Architecture

The project is structured with a modular architecture that separates concerns:



Code Flow & Data Processing Workflow

The entire process is orchestrated by `index.py` and follows these steps:

1. Scrape Item Numbers

- `item_scraper.py` retrieves all item numbers for sets, minifigures, and parts
- For each category (S, M, P), the scraper visits catalog pages and extracts item IDs
- Uses multi-threading to accelerate the process
- Returns lists of item numbers for each category

2. Download HTML Data & Images

- `data_scraper.py` takes the item numbers and downloads relevant HTML pages
- For each item, it downloads:
  - Main item page (with general information)
  - Price guide page (with pricing stats)
  - Inventory page (showing what components are in each item)
- Images are downloaded to the "Images" folder with proper filenames

- The raw HTML is stored in a SQLite database for later processing
- Uses thread pooling and retry logic for reliable downloads

3. Parse HTML Data

- `parser.py` and `parts_parser.py` extract structured data from the raw HTML
- The parsing is done in batches to optimize memory usage
- Multi-processing is used to speed up parsing with CPU cores
- For sets and minifigures, `parser.py` is used
- For parts, the specialized `parts_parser.py` is used which extracts additional color data
- The parsed data is stored in the appropriate tables in the database
- Excel reports are generated at this stage

4. Create Relationships & Finalize Database

- `db_connection.py` creates and populates a connection table
- Relationships between sets, minifigures, and parts are established
- Database indexes are created to optimize query performance
- Statistics are generated to verify data integrity
- The SQLite database is optimized for size and performance

Detailed Code File Explanation

1. index.py

Purpose: Main orchestrator script that controls the entire workflow.

Functions:

- Configures logging and parameters for each category (sets, minifigures, parts)
- Calls functions from other modules in the correct sequence
- Handles exceptions and ensures proper error reporting
- Measures overall execution time

Key aspects:

- Contains configuration for page counts and table names
- Creates output directories and database connections
- Coordinates the entire scraping, parsing, and storing process

2. item_scraper.py

Purpose: Extracts item numbers from Bricklink catalog pages.

Functions:

- `get_item()`: Extracts item numbers from a single catalog page
- `get_all_items()`: Orchestrates scraping across multiple pages using threading

Key aspects:

- Uses ThreadPoolExecutor for parallel processing
- Includes progress tracking with alive-bar
- Handles network failures and retries
- Returns clean lists of item numbers for further processing

3. data_scraper.py

Purpose: Downloads HTML content and images for each item.

Functions:

- `fetch_html()`: Retrieves HTML content from a URL
- `download_image()`: Downloads and saves images to the Images folder
- `process_item()`: Downloads all necessary data for a single item
- `process_items()`: Coordinates downloading for multiple items with threading

Key aspects:

- Includes retry logic for resilient downloading
- Custom headers to respect website policies
- Image downloading with proper referrer information
- Rate limiting to avoid overloading the server
- Stores raw HTML data in a database for later processing

4. parser.py

Purpose: General parser for sets and minifigures.

Functions:

- `separate_by_cat_type()`: Categorizes items by type
- `get_id_from_url()`: Extracts item IDs from URLs
- `extract_summary_table()`: Parses summary tables with inventory counts
- `parse_consist_of()`: Extracts component parts/minifigures/sets

- `sets_parser()`: Parses general information for sets and minifigures
- `parse_price_tables()`: Extracts pricing information
- `parse_all_data()`: Orchestrates the entire parsing process

Key aspects:

- Detailed regular expressions for accurate data extraction
- Handles various data formats and edge cases
- Multi-processing for performance with large datasets
- Converts data to database-compatible formats

5. parts_parser.py

Purpose: Specialized parser for parts with enhanced color information.

Functions:

- Contains all functions from parser.py, plus:
- `parse_color_summary_table()`: Extracts detailed color information for parts
- `parts_parser()`: Enhanced parser specifically for parts
- `parse_all_parts_data()`: Specialized orchestration for parts parsing

Key aspects:

- Extracts additional dimensions specific to parts (stud dimensions, pack dimensions)
- Captures color information including appearance type, quantity, and usage statistics
- Stores color data in a separate table for better normalization
- Generates separate Excel reports for parts and colors

6. db_connection.py

Purpose: Manages database connections, table creation, and relationships.

Functions:

- `create_connection_table()`: Creates the relationship table structure
- `populate_connection_table_batch()`: Builds relationships from parsed data
- `insert_batch()`: Handles batch insertion for performance
- `fix_wal_issue()`: Resolves SQLite WAL file issues
- `verify_statistics()`: Generates data statistics for verification
- `export_parts_colors_data()`: Exports specialized color data to Excel

Key aspects:

- Optimizes SQLite performance with pragmas
- Creates proper indexes for query optimization
- Uses batch processing to handle large datasets efficiently
- Includes detailed statistics for data verification
- Handles database maintenance operations

Database Schema

The database follows a 3rd normal form design with these main tables:

1. **set_parse**: Information about LEGO sets
   - item_number (PK), item_name, category, year_released, etc.
   - parts_relation, minifigs_relation (comma-separated lists)
2. **minifigures_parse**: Information about minifigures
   - item_number (PK), item_name, category, year_released, etc.
   - parts_relation (comma-separated list)
3. **part_parse**: Information about individual parts
   - item_number (PK), item_name, category, year_released, weight
   - dimensions, stud_dimensions, pack_dimensions, etc.
4. **part_parse_color**: Color information for parts
   - item_number (FK), appears_as, color, in_sets, total_qty
5. **connection_table**: Relationship mapping
   - id (PK), set_id (FK), minifigure_id (FK), part_id (FK)

Performance Considerations

The system is designed with several performance optimizations:

1. **Multi-threading for downloads**:
   - Parallel downloads of HTML and images using ThreadPoolExecutor
   - Configurable number of workers (default: 20)
2. **Multi-processing for parsing**:
   - CPU-intensive parsing distributed across available cores
   - Uses Pool from multiprocessing module
3. **Batch processing**:
   - Data is processed in batches (default: 1000 records)

- Reduces memory usage and allows processing large datasets

4. **Database optimizations**:
   - SQLite pragmas for improved performance
   - Proper indexes on frequently queried columns
   - Vacuum operations to reclaim space
5. **Error handling and resilience**:
   - Retry logic for network operations
   - Exception handling with proper logging
   - Transaction management for database integrity

Scraping Ethical Considerations

The scraper is designed to be respectful of the target website:

1. **Rate limiting**:
   - Throttles requests to avoid overloading the server
   - Uses appropriate delays between requests
2. **User-Agent headers**:
   - Uses legitimate user agent strings
   - Includes proper referrer information for image downloads
3. **Selective downloading**:
   - Only downloads necessary data
   - Checks if images already exist before downloading again
4. **Error handling**:
   - Gracefully handles server errors
   - Implements exponential backoff for retries

Example Output

The system generates several Excel files:

1. `parsed_sets_data.xlsx`: Complete information about LEGO sets
2. `parsed_minifigures_data.xlsx`: Detailed minifigure information
3. `parsed_parts_data.xlsx`: Comprehensive parts information
4. `parsed_parts_data_colors.xlsx`: Color-specific information for parts

Each Excel file contains all fields from the corresponding database table, organized in a readable format.

Extending the System

The modular design makes it easy to extend the system:

1. **Adding new data sources**:
   - Create a new scraper module following the pattern in item_scraper.py
   - Add the new source to the CONFIG in index.py
2. **Extracting additional fields**:
   - Enhance the parser functions in parser.py or parts_parser.py
   - Add new fields to the data dictionaries and database schema
3. **Creating new reports**:
   - Add export functions similar to export_parts_colors_data()
   - Customize the pandas DataFrame processing before export

Troubleshooting

Common issues and solutions:

1. **SQLite WAL file issues**:
   - Automatically handled by fix_wal_issue() function
   - Can be manually triggered if database files grow too large
2. **Network failures**:
   - Retry logic in data_scraper.py handles most cases
   - Adjust timeout and retry parameters for problematic connections
3. **Memory limitations**:
   - Batch size in parse_all_data() can be reduced for lower memory usage
   - Number of workers can be adjusted based on system capabilities
4. **Parsing errors**:
   - Check the logs (bricklink_scraping.log and parts_parser.log)
   - The system will continue processing other items when one fails

Conclusion

This BrickLink scraper is a comprehensive solution for collecting and organizing data about LEGO products. The modular architecture, performance optimizations, and data integrity measures make it a robust system for building a complete LEGO database. The output can be used for inventory management, price analysis, and exploring the relationships between sets, minifigures, and parts.