File: Design Standard v1.8.md

Path: Docs/Standards/Design Standard v1.8.md

Standard: AIDEV-PascalCase-1.8

Created: 2025-06-05

Last Modified: 2025-07-05 05:45PM

Design Standard v1.8

Author & Project

Author: Herb Bowers
Project: Project Himalaya
Contact: HimalayaProject1@gmail.com

Table of Contents

Purpose & Philosophy

This standard documents the unique code style, structure, and best practices for the Project Himalaya codebase.

- **Philosophy:** My code, my way—clarity, maintainability, and personality matter.
- **COD (Compulsive Order Disorder)** is a feature: consistent formatting, headers, and naming make the codebase navigable for humans, AI, and any future inheritors (post-apocalypse included).
- Where required, ecosystem and framework conventions are respected, but all other code follows these personal standards.

Header Format

ALL FILES in the project must begin with a standardized header **immediately after the shebang** (for executable scripts). This includes Python (`.py`), shell scripts (`.sh`), markdown (`.md`), text files (`.txt`), configuration files, SQL files (`.sql`), and any other project documents.

Python Files (.py)

python

```python
# File: <FileName.py>
# Path: <Full/Path/From/ProjectRoot/FileName.py>
# Standard: AIDEV-PascalCase-1.8
# Created: YYYY-MM-DD
# Last Modified: YYYY-MM-DD  HH:MM[AM|PM]
"""
Description: <Short module/class/function description>
Extended details as needed.
"""
```

Shell Scripts (.sh)

```bash
#!/bin/bash
# File: <ScriptName.sh>
# Path: <Full/Path/From/ProjectRoot/ScriptName.sh>
# Standard: AIDEV-PascalCase-1.8
# Created: YYYY-MM-DD
# Last Modified: YYYY-MM-DD  HH:MM[AM|PM]
# Description: <Short script description>
# Extended details as needed.
```

SQL Files (.sql)

```sql
-- File: <QueryName.sql>
-- Path: <Full/Path/From/ProjectRoot/QueryName.sql>
-- Standard: AIDEV-PascalCase-1.8
-- Created: YYYY-MM-DD
-- Last Modified: YYYY-MM-DD  HH:MM[AM|PM]
-- Description: <Short query/procedure/schema description>
-- Author: Herb Bowers - Project Himalaya
-- Extended details as needed.
```

Markdown/Documentation Files (.md, .txt, etc.)

```markdown
# File: <DocumentName.md>
# Path: <Full/Path/From/ProjectRoot/DocumentName.md>
# Standard: AIDEV-PascalCase-1.8
# Created: YYYY-MM-DD
# Last Modified: YYYY-MM-DD  HH:MM[AM|PM]
---
# Document Title

Description and content here.
```

Naming Conventions

Everything uses PascalCase unless ecosystem or technical requirements force exceptions.

Files & Directories

- **Python files:** `BookService.py`, `DatabaseManager.py`, `FilterPanel.py`
- **Directories:** `Source/`, `Assets/`, `Tests/`, `Scripts/`
- **Documentation:** `DesignStandard.md`, `ReadMe.md`, `MigrationGuide.md`
- **Scripts:** `UpdateFiles.py`, `CreateThumbnails.py`, `BackupDatabase.py`

Code Elements

- **Classes:** `BookService`, `DatabaseManager`, `FilterPanel`
- **Functions:** `GetCategories()`, `SearchBooks()`, `DisplayResults()`
- **Variables:** `BookTitle`, `CategoryList`, `SearchCriteria`
- **Constants:** `MAX_RESULTS`, `DEFAULT_PATH`, `API_TIMEOUT`

Database Elements

- **Databases:** `LibraryDatabase`, `UserProfiles`, `SystemLogs`
- **Tables:** `Books`, `Categories`, `UserSessions`, `AuditLogs`
- **Columns:** `BookTitle`, `CategoryName`, `CreatedDate`, `LastModified`
- **Indexes:** `IX_Books_Category`, `IX_Users_Email`, `IX_Logs_Date`
- **Constraints:** `PK_Books_ID`, `FK_Books_Category`, `UK_Users_Email`
- **SQL Variables:** `@BookCount`, `@CategoryFilter`, `@StartDate`
- **SQL Aliases:** `B` (for Books), `C` (for Categories), `U` (for Users)
- **Procedures:** `GetBooksByCategory`, `UpdateUserPreferences`
- **Triggers:** `TR_Books_UpdateTimestamp`, `TR_Users_LogChanges`

Ecosystem Exceptions

- **Python packages:** `__init__.py`, `setup.py` (required by Python)
- **Web files:** Lowercase when required by web standards
- **Third-party tools:** Follow tool conventions when necessary (documented in header)

Complete SQL Example:

sql

```sql
-- Good: Full PascalCase compliance
SELECT B.BookTitle, C.CategoryName, B.Rating
FROM Books B
  INNER JOIN Categories C ON B.CategoryID = C.CategoryID
WHERE B.CreatedDate >= @StartDate
  AND C.CategoryName LIKE @CategoryFilter
ORDER BY B.BookTitle;

-- Avoid: Traditional snake_case
SELECT b.book_title, c.category_name, b.rating
FROM books b
  INNER JOIN categories c ON b.category_id = c.category_id
WHERE b.created_date >= @start_date;
```

Design Standards

Note: These standards apply to all production code. Exception: 1-shot down and dirty scripts may deviate from these requirements when documented.

Code Organization

- **Module size limit:** No module should exceed 300 lines of code
- **Single responsibility:** Modules should address unique sets of design elements
- **Cohesion:** Related functionality should be grouped together
- **Coupling:** Minimize dependencies between modules

Database Design Principles

- **Normalization:** Databases should be normalized but not at excessive levels (typically 3NF, avoid over-normalization)
- **Change tracking:** Primary tables should track user changes (CreatedBy, CreatedDate, LastModifiedBy, LastModifiedDate)
- **Portability:** Build with consideration of porting to more sophisticated database engines (PostgreSQL, SQL Server)
- **Performance:** Maximize the use of tables and proper indexing to enhance access times
- **Audit trail:** Maintain comprehensive logging of data modifications

Development Practices

- **Modularity:** Design for reusability and maintainability
- **Documentation:** Every design decision should be documented
- **Testing:** Design with testability in mind from the start
- **Scalability:** Consider future growth and performance requirements

---

File & Directory Structure

- **Directory tree** documented at project root; updated as project evolves.
- **Directory names:** `PascalCase` unless system conventions require otherwise (e.g., `.git`, `node_modules`)
- Each directory can have a `README.md` summarizing its contents and purpose.
- Test files in `/Tests` directory, following header and naming conventions.

Standard Project Directory Structure

```
.
├── ./Assets              # Static assets (images, icons, etc.)
├── ./Source              # Main source code (PascalCase)
│   ├── ./Core            # Business logic and services
│   ├── ./Data            # Data models and database access
│   ├── ./Interface       # UI components and windows
│   ├── ./Utils           # Utility functions and helpers
│   └── ./Framework       # Reusable framework components
├── ./Tests               # Unit tests and test data
├── ./Scripts             # Deployment and utility scripts
├── ./Docs                # All documentation
│   ├── ./Standards       # Design standards and guidelines
│   ├── ./Architecture    # System architecture docs
│   ├── ./Updates         # Update logs and reports
│   └── ./Daily           # Daily development notes
├── ./Archive             # Archived versions of files
├── ./Updates             # Temporary folder for file updates
├── ./Legacy              # Legacy code being phased out
└── ./Assets              # Static resources and data files
```

Project Setup Standards

- **Automated setup scripts required** for all new environments
- **`requirements.txt`** or `pyproject.toml` for Python dependencies

- **Environment validation** on startup with clear error messages
- **Standard `.gitignore`** template used across all repositories
- **Database initialization** scripts for clean setup

Standard .gitignore Template

```gitignore
# Python
__pycache__/
*.pyc
*.pyo
*.egg-info/
.pytest_cache/

# Environment
.env
.venv/
venv/

# IDE
.vscode/
.idea/
*.swp
*.swo

# OS
.DS_Store
Thumbs.db

# Project-specific
*.log
temp/
cache/
```

Automated File Management

Critical Workflow: The UpdateFiles.py script automates Design Standard v1.8 compliance and file management, eliminating manual work and ensuring consistency.

Purpose & Benefits

- **Automated compliance:** PascalCase enforcement and header validation

- **Streamlined updates:** Drop files in `Updates/` folder and run script
- **Complete audit trail:** Full logging of all file operations
- **Backup protection:** Automatic archiving with timestamps
- **Error prevention:** Eliminates manual copy mistakes

File Preparation for Updates

ALL files intended for the update system MUST include a proper `Path:` header that specifies the destination relative to project root:

```python
# File: BookService.py
# Path: Source/Core/BookService.py
# Standard: AIDEV-PascalCase-1.8
# Created: 2025-07-05
# Last Modified: 2025-07-05  05:31PM
```

```sql
-- File: CreateUsersTable.sql
-- Path: Scripts/Database/CreateUsersTable.sql
-- Standard: AIDEV-PascalCase-1.8
-- Created: 2025-07-05
-- Last Modified: 2025-07-05  05:31PM
```

```markdown
# File: MigrationGuide.md
# Path: Docs/Architecture/MigrationGuide.md
# Standard: AIDEV-PascalCase-1.8
# Created: 2025-07-05
# Last Modified: 2025-07-05  05:31PM
```

Update Workflow

1. **Preparation:** Place updated files in `/Updates` folder with proper headers
2. **Execution:** Run `python UpdateFiles.py` from project root
3. **Automation:** Script reads `Path:` headers and moves files to correct locations
4. **Backup:** Existing files automatically archived with timestamps to `/Archive`
5. **Compliance:** All paths and filenames converted to PascalCase

6. **Audit:** Complete status report generated in `/Docs/Updates`

Script Capabilities

- **Header parsing:** Extracts destination path from `Path:` header in any file type
- **Base directory stripping:** Removes known base directories (ProjectHimalaya, BowersWorld-com)
- **PascalCase enforcement:** Converts all paths and filenames to Design Standard v1.8
- **Archiving:** Moves existing files to timestamped archive before replacement
- **Documentation handling:** Moves `.md` and `.txt` files to dated documentation folders
- **Error handling:** Comprehensive logging and graceful failure recovery
- **Status reporting:** Detailed markdown report with success/failure statistics

Example Update Session

```bash
# Place files in Updates folder
Updates/
├── FilterPanel.py       # Path: Source/Interface/FilterPanel.py
├── BookGrid.py          # Path: Source/Interface/BookGrid.py
├── MainWindow.py        # Path: Source/Interface/MainWindow.py
└── BookService.py       # Path: Source/Core/BookService.py

# Run update script
python UpdateFiles.py

# Results:
# ✅ 4 files moved successfully
# ✅ 4 existing files archived
# ✅ All paths converted to PascalCase
# ✅ Audit report: Docs/Updates/Updates_2025-07-05_17-31-25.md
```

Integration with AI Development

- **Prepare files with proper headers:** AI can generate files with correct `Path:` headers
- **Bulk updates:** Multiple files can be processed in single update session
- **Version control friendly:** Automatic archiving preserves development history
- **Standards enforcement:** Impossible to accidentally violate naming conventions
- **Audit compliance:** Every change tracked and documented

This automated system makes Design Standard v1.8 compliance effortless and eliminates the maintenance overhead that would otherwise make the standard impractical.

Development Environment

Standard Environment

- **OS:** Ubuntu 25.04 (primary), Windows 11 (secondary)
- **IDE:** VS Code with Python extension
- **Python:** 3.11+ with virtual environments
- **Hardware:** AMD Ryzen 7 5800X, 32GB RAM, RTX 3070

Required Tools

- **Git:** Version control with proper commit messaging
- **Virtual Environment:** `python -m venv` for isolation
- **Package Management:** `pip` with `requirements.txt`
- **Testing:** `pytest` for unit testing framework
- **Code Quality:** `pylint` or `flake8` for linting

Imports & Dependencies

Import Organization

python

```python
# Standard library imports
import sys
import os
import logging
from pathlib import Path
from typing import List, Optional, Dict

# Third-party imports
import PySide6
from PySide6.QtWidgets import QWidget, QVBoxLayout
import sqlite3

# Local imports
from Source.Core.DatabaseManager import DatabaseManager
from Source.Data.DatabaseModels import Book
```

Guidelines

- **Group imports** by category (standard, third-party, local)
- **Alphabetical order** within each group
- **Multi-line imports:** Each import on its own line.
- **Use** `isort` (optional) for automation.
- **Dependencies:** Centralized in `requirements.txt` or `pyproject.toml`.

Coding Style & Documentation

- **PEP8** is respected where it does not conflict with these standards.
- **Type hints** are strongly encouraged for all public functions.
- **All functions/classes** must have docstrings.
- **Minimum comment level:** All non-trivial logic is commented for intent.
- **Error handling:** Use `try/except` with clear logging, fail early if possible. Custom exceptions as needed.
- **Logging:** Prefer Python's `logging` module over print statements.

Testing & Quality

- **All code must be covered by `pytest` unit tests.**

- **Test coverage goal:** 80%+
- **Test files follow header standard.**
- **Test data** (e.g., sample PDFs) stored in `/Tests/Data` with README as needed.
- **Performance/benchmark tests** included for GPU/CPU code as appropriate.

SQL and Data Access

- **NO SQLAlchemy.**
  - Use raw SQL and parameterized queries only.
  - SQLite is default.
  - PostgreSQL/SQL Server for production when needed.
- **Database naming:** PascalCase for ALL elements (tables, columns, indexes, constraints)
- **SQL file naming:** `CreateUserProfilesTable.sql`, `UpdateSchema_v1_2.sql`
- **SQL files must use standard headers** with File, Path, Standard, Created, Last Modified, Description, and Author fields.

Note: This comprehensive PascalCase approach maintains complete visual consistency throughout the entire technology stack while remaining compatible with all major SQL engines (SQLite, PostgreSQL, MySQL, SQL Server).

Third-Party Libraries & Ecosystem Exceptions

- **Where frameworks require specific conventions** (pytest, Flask, Django, etc.), those are followed and noted in file header with justification.
- **Special files** like `__init__.py`, `setup.py`, and `test_*.py` are exempt from PascalCase rule when tools explicitly require snake_case.
- **Web standards** that require lowercase (e.g., certain HTML/CSS files) are exempt when technical requirements mandate it.
- **Other third-party quirks** are documented inline and in module README if needed.
- **All exceptions must be justified** in the file header under "Exception Reason."

AI Collaboration Practices

- Major changes generated or reviewed by AI (ChatGPT, Claude, etc.) are noted in the header or docstring.
- AI-generated refactoring/design is tracked via comments or commit messages for transparency.

- All contributors (human or AI) are acknowledged in the attribution section.
- **File updates for AI:** Use proper `Path:` headers for automated update system integration.

Attribution & License

- Attribution and contact are included at the head of the standard and in each major module as needed.
- **License:** (insert your preferred open source license here, e.g., MIT, Apache 2.0)
- Special thanks to the open-source community and the AI models that help build and document this project.

Revision History

- **1.6:** Original AIDEV-PascalCase Standards (Herb Bowers)
- **1.7:**
  - Clarified ecosystem exceptions (special files, third-party libs)
  - Formalized "No SQLAlchemy" policy
  - Added sections on project structure, testing, and attribution
  - Baked in session-based clarifications and "Himalaya Addenda"
  - Updated header example and philosophy notes
- **1.8:**
  - **Extended PascalCase to ALL database elements** (databases, tables, columns, indexes, constraints)
  - **Mandated standardized headers for ALL file types** (.py, .sh, .md, .txt, config files, etc.)
  - **Emphasized critical importance of updating "Last Modified" timestamps**
  - **Clarified filename PascalCase rules with specific exceptions**
  - **Added comprehensive Design Standards section** (300-line module limit, database design principles)
  - **Defined standard project directory structure** (Assets, Source, Scripts, Tests, etc.)
  - **Added Project Setup Standards** (automated setup script requirements)
  - **Documented standard development environment** (Ubuntu 25.04, VS Code, hardware specs)
  - **Provided standard .gitignore template** with project-specific exclusions
  - Updated directory naming to PascalCase (`/Tests` instead of `/tests`)

- Added comprehensive examples for different file type headers
- **COMPREHENSIVE SQL NAMING STANDARDS:** Extended PascalCase to ALL SQL elements including indexes, constraints, variables, aliases, procedures, and triggers with complete elimination of underscores
- **Added SQL file header requirements** and examples
- **Provided detailed SQL naming examples** showing correct vs. incorrect patterns
- **NEW: Added Automated File Management section** documenting UpdateFiles.py workflow, header requirements, and integration with AI development processes

---

This standard is a living document. Updates are versioned, and the latest version governs all code, docs, and scripts for Project Himalaya. For changes, contact the author.