Guide for an AI Agent Automating sdAIgenLightning File Population

This guide outlines the architecture and specific formatting rules within the sdAIgenLightning project, enabling an AI agent to programmatically add and manage models, VAEs, LoRAs, ControlNets, and other assets.

1. Project Architecture and Data Flow Overview

The sdAIgenLightning project leverages a set of Python scripts to manage Stable Diffusion WebUI environments. The core idea for data population is:

1. **Data Definition (scripts/_*-data.py):** Python dictionaries define lists of models, LoRAs, VAEs, and ControlNets, specifying their URLs and desired local filenames.
2. **Widget Interface (scripts/widgets-en.py):** This script reads the data definition files and dynamically populates interactive dropdown menus in the Jupyter Notebook interface.
3. **Download Orchestration (scripts/downloading-en.py):** Based on user selections from the widgets (or direct input), this script constructs download commands, resolves URLs, handles CivitAI API interactions, and initiates the actual file downloads.
4. **Core Download Logic (modules/Manager.py):** This module provides the underlying functions for downloading files from various sources (Hugging Face, CivitAI, Google Drive, GitHub) and performing basic file operations.
5. **CivitAI Integration (modules/CivitaiAPI.py):** Specifically handles resolution of CivitAI model page URLs to direct download links and fetches associated metadata (like preview images).
6. **Path Management (modules/webui_utils.py):** Centralizes the definition of file system paths, particularly pointing all model-related assets to a shared storage location (sd_models_shared).
7. **Environment Setup (scripts/setup.py):** Initializes the notebook environment, downloads core project files (including all scripts and modules), and sets up platform-aware paths.
8. **Launch (scripts/launch.py):** Configures and launches the selected Stable Diffusion WebUI, ensuring it points to the correct model directories.

For an AI agent, the primary interaction points for automation will be:

- **Modifying the _*-data.py files:** To add new predefined options to the widgets.
- **Generating input for downloading-en.py:** To trigger custom downloads programmatically, bypassing direct widget interaction.
- **Understanding the final file destinations:** To verify successful downloads or manage files post-download.

2. Relevant File Structure

Here's a visualization of the most relevant file structure within the drf0rk/sdAIgenLightning

repository that pertains to populating widgets and managing downloads:

```
drf0rk/sdAIgenLightning/
├── scripts/
│   ├── _loras-data.py          # LoRA model definitions (SD 1.5 and SDXL)
│   ├── _models-data.py         # SD 1.5 model, VAE, ControlNet definitions
│   ├── _xl-models-data.py      # SDXL model, VAE, ControlNet definitions
│   ├── launch.py               # Launches the selected WebUI with configured paths
│   ├── setup.py                # Initial setup, downloads project files from fork
│   ├── download-result.py      # Displays downloaded files (for verification/cleanup)
│   ├── cleaner_gui.py          # GUI for cleaning up project folders
│   ├── auto-cleaner.py         # Script for automated cleaning
│   ├── en/                     # English language-specific scripts
│   │   ├── downloading-en.py   # Orchestrates downloads based on selections
│   │   └── widgets-en.py       # Dynamically populates widgets from data files
│   └── ru/                     # Russian language-specific scripts (similar to en/)
│       ├── downloading-ru.py
│       └── widgets-ru.py
├── modules/
│   ├── CivitaiAPI.py           # Handles CivitAI URL resolution and metadata fetching
│   ├── Manager.py              # Core download functions (aria2c, gdown, curl, git clone)
│   ├── webui_utils.py          # Manages WebUI paths, centralizes shared model base
│   ├── json_utils.py           # Utility for JSON file operations (settings.json)
│   ├── widget_factory.py       # Helper for creating ipywidgets UI elements
│   ├── __season.py             # Displays environment info
│   └── TunnelHub.py            # Manages tunneling services for WebUI access
├── __configs__/                # Configuration files for different UIs and general settings
│   ├── A1111/
│   │   ├── config.json
│   │   └── ui-config.json
│   ├── ComfyUI/
│   │   ├── comfy.settings.json
│   │   ├── install-deps.py
│   │   └── workflows/
│   │       └── anxety-workflow.json
│   ├── Classic/
│   │   ├── config.json
│   │   └── ui-config.json
│   ├── Forge/
│   │   ├── config.json
│   │   └── ui-config.json
│   ├── ReForge/
│   │   ├── config.json
│   │   └── ui-config.json
```

```
│   ├── SD-UX/
│   │   ├── config.json
│   │   └── ui-config.json
│   ├── gradio-tunneling.py
│   ├── styles.csv              # Predefined text styles
│   └── user.css                # Custom CSS for the UI
├── notebook/
│   ├── ANXETY_sdAIgen_EN.ipynb    # The main English Jupyter Notebook
│   └── ANXETY_sdAIgen_RU.ipynb    # The main Russian Jupyter Notebook
└── CSS/
│   ├── auto-cleaner.css
│   ├── download-result.css
│   └── main-widgets.css
└── JS/
    └── main-widgets.js
```

3. Data Definition: _*-data.py Files (The Source)

These Python files (_models-data.py, _loras-data.py, _xl-models-data.py) define the lists of assets that appear in the widget dropdowns. They use a standard Python dictionary structure.

Structure Example:

```
# scripts/_models-data.py
## MODEL

model_list = {
    "1. Anime (by XpucT) + INP": [
        {'url': "https://huggingface.co/XpucT/Anime/resolve/main/Anime_v2.safetensors", 'name':
"Anime_V2.safetensors"},
        {'url':
"https://huggingface.co/XpucT/Anime/resolve/main/Anime_v2-inpainting.safetensors", 'name':
"Anime_V2-inpainting.safetensors"}
    ],
    "2. BluMix [Anime] [V7] + INP": [
        {'url': "https://civitai.com/api/download/models/361779", 'name': "BluMix_V7.safetensors"},
        {'url': "https://civitai.com/api/download/models/363850", 'name':
"BluMix_V7-inpainting.safetensors"}
    ]
    # ... more entries
}

## LORA (in scripts/_loras-data.py)

lora_data = {
```

```
    "sd15_loras": {
        "1. Example SD1.5 LoRA A": [
            {'url': "https://civitai.com/api/download/models/YOUR_SD15_LORA_ID_A", 'name':
"sd15_lora_a.safetensors"}
        ],
    },
    "sdxl_loras": {
        "1. nudify-xl": [
            {'url': "https://civitai.com/models/9025?modelVersionId=177674", 'name':
"nudify_xl_lite.safetensors"}
        ],
    }
}
```

Key Formatting Rules for AI Agent:

- **Top-Level Keys:** Human-readable strings that appear in the dropdowns (e.g., "1. Anime (by XpucT) + INP"). It's good practice to keep the numbering for consistency.
- **Value (List of Dictionaries):** Each top-level key maps to a Python list of dictionaries. Each dictionary represents a file to be downloaded as part of that selection.
- **Inner Dictionary ({'url': '...', 'name': '...'}):**
    - **'url' (String):** The primary source URL for the asset. This can be:
        - **Direct Download Link:** (e.g., from Hugging Face https://huggingface.co/.../resolve/... or CivitAI API https://civitai.com/api/download/models/...). This is the most straightforward.
        - **CivitAI Model Page URL:** (e.g., https://civitai.com/models/MODEL_ID?modelVersionId=MODEL_VERSION_ID). The CivitaiAPI.py module will automatically resolve this to a direct download link and fetch metadata. This is a very powerful feature for the agent.
        - **Google Drive Link:** These are handled by gdown via Manager.py.
    - **'name' (String, Optional):** This specifies the exact filename for the downloaded asset on the local file system (e.g., "Anime_V2.safetensors").
        - **Recommendation: Always include the 'name' key** with the desired filename (including the correct extension like .safetensors, .ckpt, .pt, .yaml, etc.). This gives the agent precise control over naming.
        - **If omitted:** The system will attempt to derive a filename from the URL (e.g., taking the last segment of the URL path). For CivitAI model page URLs, CivitaiAPI will provide a default filename based on the model's metadata.

AI Agent Action: To add a new model option, the agent would need to:

1. Read the content of the relevant _*-data.py file.
2. Parse its Python dictionary structure.
3. Add a new key-value pair to the dictionary, following the specified format. Ensure the URL is valid and the name is explicitly set with the correct extension.

4. Serialize the modified dictionary back into Python code and overwrite the _*-data.py file. (Caution: Parsing and re-serializing Python code can be complex. A more robust solution for large-scale automation might involve storing this data in JSON and modifying the read_model_data function to load JSON instead of exec()ing Python files, but this is a significant architectural change).

4. Widget Population: scripts/widgets-en.py (The Display)

This script dynamically generates the interactive widgets.

Key Mechanism: read_model_data()

```
# Excerpt from scripts/en/widgets-en.py
def read_model_data(file_path, data_key_in_file, prefixes=['none']):
    """Reads data from the specified file based on a key within that file."""
    local_vars = {}
    with open(file_path) as f:
        exec(f.read(), {}, local_vars) # DANGER: Executes the data file as Python code!

    # Safely get the nested dictionary if data_key_in_file is like 'lora_data.sd15_loras'
    data_dict = local_vars
    for key_part in data_key_in_file.split('.'):
        data_dict = data_dict.get(key_part, {})
        if not isinstance(data_dict, dict):
            data_dict = {}
            break

    names = list(data_dict.keys())
    return prefixes + names
```

- read_model_data directly exec()utes the content of the _*-data.py files. This means any changes made to _*-data.py by the AI agent will be picked up when widgets-en.py is next run.
- The data_key_in_file argument allows specifying nested keys, e.g., 'lora_data.sd15_loras' to access specific parts of the lora_data dictionary.
- The function returns a list of keys (the human-readable names) which then populate the dropdown options.

Dynamic Widget Updates (XL_models_widget):

- The XL_models_widget (SDXL checkbox) has a callback (update_XL_options) that is crucial. When its value changes, it dynamically updates the options for the model_widget, vae_widget, controlnet_widget, and lora_widget by calling read_model_data again with either _models-data.py / _loras-data.py (for SD 1.5) or _xl-models-data.py / _loras-data.py (sdxl_loras key) (for SDXL).
- **AI Agent Action:** If the AI agent intends to use SDXL models, it needs to ensure this checkbox's state is set to True either programmatically before downloading-en.py runs, or by understanding that _xl-models-data.py and lora_data.sdxl_loras contain the

relevant entries.

5. Download Logic: scripts/en/downloading-en.py & modules/Manager.py (The Engine)

This is where the rubber meets the road for file population. The downloading-en.py script orchestrates downloads, and Manager.py performs them.

5.1. The PREFIX_MAP and Target Directories

A critical component is the PREFIX_MAP in downloading-en.py:

```
# Excerpt from scripts/en/downloading-en.py
PREFIX_MAP = {
    # prefix : (dir_path , short-tag)
    'model': (model_dir, '$ckpt'),
    'vae': (vae_dir, '$vae'),
    'lora': (lora_dir, '$lora'),
    'embed': (embed_dir, '$emb'),
    'extension': (extension_dir, '$ext'),
    'adetailer': (adetailer_dir, '$ad'),
    'control': (control_dir, '$cnet'),
    'upscale': (upscale_dir, '$ups'),
    # Other (mostly ComfyUI specific)
    'clip': (clip_dir, '$clip'),
    'unet': (unet_dir, '$unet'),
    'vision': (vision_dir, '$vis'),
    'encoder': (encoder_dir, '$enc'),
    'diffusion': (diffusion_dir, '$diff'),
    'config': (config_dir, '$cfg')
}
```

- **Purpose:** This dictionary maps content prefix (like 'model', 'lora') to their corresponding local directory (dir_path) and an optional short-tag (used in Empowerment mode).
- **dir_path Resolution:** The dir_path values (e.g., model_dir, lora_dir) are obtained from the settings.json file, which is populated by webui_utils.py. All model-related dir_paths resolve to subdirectories within a centralized SHARED_MODEL_BASE (e.g., sd_models_shared/Stable-diffusion, sd_models_shared/loras). This is important for an AI agent to know where files will be stored. Extensions are generally saved in the UI-specific extensions folder.

5.2. Input Formatting for download(line)

The download(line) function in downloading-en.py is the workhorse. It accepts a single string (line) containing comma-separated download instructions.

AI Agent Action: The agent can construct this line string directly to initiate custom

downloads.

Formats understood by download(line):

1. **Widget-Generated Format (from handle_submodels):**
    - **Format:** URL DESTINATION_PATH FILENAME (space-separated)
    - **Example:**
      https://huggingface.co/XpucT/Anime/resolve/main/Anime_v2.safetensors /root/sd_models_shared/Stable-diffusion Anime_V2.safetensors
    - **Notes:**
        - DESTINATION_PATH: Must be the full absolute path to the target directory (e.g., /root/sd_models_shared/Stable-diffusion).
        - FILENAME: Must be the exact desired filename, including extension.
        - handle_submodels function within downloading-en.py generates this format based on widget selections.
2. Custom Download / Empowerment Mode (empowerment_output_widget input):
   This is the most flexible format for an AI agent to specify downloads directly.
    - **Format:** prefix:URL[filename.ext] or prefix:URL
    - **Components:**
        - **prefix:**: One of the keys from PREFIX_MAP (e.g., model, vae, lora, extension, control, etc.). This tells the script *where* to save the file.
            - Example: lora:, extension:, model:.
        - **URL**: The actual download URL (Hugging Face, CivitAI API, CivitAI model page, etc.).
        - **[filename.ext] (Optional, enclosed in square brackets []):** This explicitly specifies the desired filename (including its extension) for the downloaded file.
            - **Recommendation for AI Agent: Always use this [filename.ext] tag** to ensure precise naming, especially for extensions or files where the URL doesn't clearly indicate the desired name or extension.
            - Example: [my_custom_lora.safetensors], [Regional-Prompter].
        - **Examples for AI Agent to Construct:**
            - Adding a new LoRA:
              lora:https://civitai.com/api/download/models/123456[my_new_lora.safetensors]
            - Adding a new Extension:
              extension:https://github.com/some-user/some-extension[Some-Extension-Repo]
              (Note: for extensions, the [filename] is the repository name, which Manager.py uses for git clone.)
            - Adding a new Model (using a CivitAI Model Page URL):
              model:https://civitai.com/models/7890?modelVersionId=101112[my_awesome_model.safetensors]
              (The CivitaiAPI will handle resolving this to the direct download link and validating the filename.)

        - Adding a new VAE: vae:https://huggingface.co/username/repo/resolve/main/my_vae.safetensors[custom_vae.safetensors]
    - **Multiple Downloads:** Separate multiple download instructions with a comma or space within the same line string.
        - Example: lora:URL1[name1], model:URL2[name2] extension:URL3[name3]

5.3. URL Cleaning and Filename Extraction (_clean_url, _extract_filename)

These functions within downloading-en.py (and Manager.py) are crucial for processing URLs:

- **_clean_url(url):**
    - Transforms Hugging Face blob URLs to resolve URLs.
    - Transforms GitHub blob URLs to raw URLs.
    - **Crucially, for CivitAI URLs, it calls CivitaiAPI.validate_download(url)** to get the actual direct download link and validated metadata. This means the AI agent can safely provide CivitAI *model page URLs* (not just direct download links) to the system, and it will resolve them correctly.
- **_extract_filename(url):**
    - First attempts to find and extract a filename enclosed in square brackets [] within the URL string. This is the **highest precedence** for specifying a custom filename.
    - If no [] tag is found, and it's a CivitAI or Google Drive URL, it returns None, indicating that the filename will be determined by the CivitaiAPI or gdown tool respectively.
    - Otherwise, it uses Path(urlparse(url).path).name (the last segment of the URL path) as the default filename.

5.4. The Actual Download (manual_download and download_file_platform_aware)

- manual_download in downloading-en.py prepares the download. It uses CivitaiAPI.py for CivitAI URLs (which also fetches preview images and their names).
- It then calls download_file_platform_aware (which uses requests and tqdm for progress) to perform the actual file transfer.
- **Post-Download:** _unpack_zips() is called after all downloads. If any downloaded file is a .zip archive, it will be automatically extracted, and the .zip file will be deleted.

5.5. Git Cloning for Extensions

- When extension: prefix is used, the URL and (optional) repository name are added to extension_repo.
- After all other downloads, downloading-en.py iterates through extension_repo and uses git clone --depth 1 --recursive to clone the repositories into the extension_dir (which is configured in settings.json by webui_utils.py to be UI-specific, e.g., A1111/extensions).
- **AI Agent Action:** When adding extensions, provide the GitHub repository URL and, optionally, a custom folder name in [ ] if the default repo name is undesirable.

6. Persistent Storage and Path Management: modules/webui_utils.py

This module is crucial for understanding *where* the downloaded files ultimately reside.

- **SHARED_MODEL_BASE = HOME / 'sd_models_shared'**: This is the central, platform-aware directory for all model-related assets. It ensures models are stored efficiently and consistently across different WebUI installations (A1111, ComfyUI, Forge, etc.).
- **_set_webui_paths(ui):** This function, called during setup, sets up the correct subdirectories under SHARED_MODEL_BASE (e.g., sd_models_shared/Stable-diffusion, sd_models_shared/loras, sd_models_shared/vae). It also sets the UI-specific extension_dir and output_dir.
- **AI Agent Action:** When planning to download or verify files, the agent should understand that files like checkpoints (.safetensors, .ckpt), LoRAs (.safetensors), VAEs (.safetensors), embeddings (.pt, .safetensors), ControlNet models (.safetensors, .yaml), etc., will be placed in their respective subdirectories within SHARED_MODEL_BASE. Extensions, however, go into the UI-specific extensions folder.

7. Notebook Integration: ANXETY_sdAIgen_EN.ipynb (The Orchestrator)

The Jupyter Notebook (ANXETY_sdAIgen_EN.ipynb) acts as the top-level orchestrator.

- **%run Command:** The notebook uses the %run magic command to execute Python scripts (setup.py, widgets-en.py, downloading-en.py, launch.py) sequentially.
- **setup.py's Role:**
    - It's the very first script executed.
    - It detects the platform (Colab, Kaggle, Lightning AI, Local).
    - It downloads all the core project files (CSS, JS, modules, scripts) from the specified GitHub fork (drf0rk/sdAIgenLightning in your case). This means any changes made by the AI agent to the data files (_*-data.py) or logic scripts (downloading-en.py, widgets-en.py) *within your fork* will be deployed to the runtime environment.
    - It sets up the ANXETY folder as SCR_PATH and adds SCR_PATH/modules to sys.path, making all utility modules importable.
    - It defines SHARED_MODEL_BASE and saves platform-aware paths to settings.json.

AI Agent Workflow for Full Automation:

1. **Modify Data Files (Offline/Pre-Run):** The AI agent would first modify the _models-data.py, _loras-data.py, _xl-models-data.py files in its local copy of the drf0rk/sdAIgenLightning repository (or generate them directly if it's creating a new project). These changes would then need to be pushed to the GitHub repository that the notebook is pulling from.
2. **Run setup.py:** The agent would initiate the execution of the first cell in the notebook (%run setup.py). This ensures the latest versions of all scripts (including the modified data files) are pulled into the environment.
3. **Configure widgets-en.py Parameters:** If the agent wants to control which

models/LoRAs are downloaded via the "standard" widget selection, it can set the corresponding Python variables before executing the widgets-en.py cell. For example, to download a specific model from _models-data.py:
# Before %run $scripts_dir/$lang/widgets-{lang}.py
model = '2. BluMix [Anime] [V7] + INP' # Select this model by its key
model_num = '' # No numerical selection needed
XL_models = False # Ensure SD 1.5 models are selected
# ... other widget defaults

Alternatively, for custom downloads, the agent would construct the empowerment_output_widget.value string.

4. **Run widgets-en.py:** Execute the widgets cell to initialize the UI and load the model data.
5. **Trigger Download (downloading-en.py):** Execute the downloading cell. This will use the values set in the previous step (or custom inputs) to trigger the file downloads.
6. **Launch WebUI (launch.py):** Execute the launch cell to start the Stable Diffusion WebUI. The WebUI will automatically find the downloaded assets because the paths have been correctly configured by webui_utils.py and launch.py.

Conclusion

Automating file population in sdAIgenLightning for an AI agent involves a two-pronged approach:

1. **Direct Manipulation of Data Definition Files:** Modifying _*-data.py to add new static options for models, LoRAs, etc., which are then picked up by widgets-en.py via exec(). Remember to provide explicit url and name (with extension) for each asset.
2. **Programmatic Input to the Download Script:** Directly constructing the input string for downloading-en.py (especially using the prefix:URL[filename.ext] format) to dynamically download assets not present in the predefined lists. This offers the most flexibility.

Always ensure that your AI agent understands the target file paths as defined by webui_utils.py (especially SHARED_MODEL_BASE) to correctly manage and verify downloads. By following these formatting rules and understanding the project's data flow, your AI agent can effectively automate the population of models and other assets within this system.