Developer Guide - YouTube Shorts Creator Windows

Overview

This document provides technical details for developers who want to understand, modify, or contribute to the YouTube Shorts Creator Windows application.

Architecture

Component Structure

```
┌──────────────────────────────────────────────┐
│         shorts_creator_gui.py                │
│                                              │
│  - Tkinter GUI Interface                     │
│  - User Input Validation                     │
│  - Threading for Background Processing       │
│  - Console Output Redirection                │
│  - Progress Indication                       │
└──────────────────────────────────────────────┘
               │
               │
               │ imports & calls
               │
               ▼
┌──────────────────────────────────────────────┐
│       shorts_creator_core.py                 │
│                                              │
│  - Video Processing Logic                    │
│  - MoviePy Integration                       │
│  - Whisper Speech-to-Text                    │
│  - Caption Generation                        │
│  - Audio Mixing                              │
└──────────────────────────────────────────────┘
```

Key Classes

`ShortsCreatorGUI` (shorts_creator_gui.py)

Main GUI application class.

Responsibilities:
- Create and manage Tkinter widgets
- Handle user input and file selection
- Validate inputs before processing
- Manage background thread for video processing
- Display progress and console output
- Handle errors gracefully

Key Methods:
- `__init__(root)` : Initialize GUI and variables
- `_create_widgets()` : Build the UI

- `_validate_inputs()` : Check all inputs are valid
- `_create_short()` : Start video creation in background thread
- `_process_video()` : Background worker method (runs in thread)
- `_processing_complete()` : Handle completion or errors

`ShortsCreator` (shorts_creator_core.py)

Core video processing engine.

Responsibilities:
- Load and process video/audio files
- Create split-screen layout
- Generate captions (manual or auto)
- Mix audio tracks
- Export final video

Key Methods:
- `create_short()` : Main orchestration method
- `_transcribe_audio()` : Use Whisper for speech-to-text
- `_chunk_words()` : Group words into readable caption segments
- `_resize_and_crop()` : Fit videos to 9:16 aspect ratio
- `_create_static_caption()` : Generate manual caption overlay
- `_create_auto_captions()` : Generate dynamic auto-captions

Technical Details

Video Processing Pipeline

```
1. Load Videos
   ├─ Original video (top)
   └─ Reaction video (bottom)

2. Sync Duration
   └─ Trim both to shortest duration

3. Layout Creation
   ├─ Resize original to 60% height (1080x1092)
   ├─ Resize reaction to 40% height (1080x708)
   ├─ Create black divider bar (1080x120)
   └─ Position all elements

4. Caption Generation (if enabled)
   ├─ Manual: Create static text overlay
   └─ Auto: Transcribe audio → Generate timed captions

5. Audio Mixing
   ├─ Keep reaction audio (100%)
   ├─ Add background music (30%)
   └─ Mute original video

6. Composite & Export
   └─ Combine all layers → Export as MP4
```

Dependencies

Core Dependencies

- **moviepy 1.0.3**: Video editing and composition
- **openai-whisper**: Speech-to-text transcription
- **numpy**: Array operations for video data
- **Pillow**: Image processing for text rendering
- **imageio-ffmpeg**: FFmpeg wrapper for video encoding

GUI Dependencies

- **tkinter**: Built-in Python GUI framework (no install needed)

Packaging Dependencies

- **pyinstaller**: Convert Python to executable

Whisper Models

Model	Size	Speed	Accuracy	RAM Usage	Disk Space
tiny	39M	Fastest	Good	~1GB	~100MB
base	74M	Fast	Better	~1GB	~150MB
small	244M	Medium	Great	~2GB	~500MB
medium	769M	Slow	Excellent	~5GB	~1.5GB
large	1550M	Very Slow	Best	~10GB	~3GB

Building the Executable

Prerequisites

```
# Python 3.8+
python --version

# Install dependencies
pip install -r requirements.txt
```

Build Options

Option 1: Folder-Based Build (Recommended)

Creates a folder with the executable and dependencies.

```
# Windows
build_windows.bat

# Linux (for testing)
./build_linux.sh
```

Pros:
- Faster startup time
- Easier to debug
- Better for distribution

Cons:
- Multiple files to distribute

Option 2: Single-File Build

Creates one large executable.

```
pyinstaller build_onefile.spec --clean
```

Pros:
- Single file to distribute

Cons:
- Slower startup (extracts files to temp)
- Larger file size (~500MB)
- Some antivirus software may flag it

Build Configuration

Key files:
- `build.spec`: Folder-based build configuration
- `build_onefile.spec`: Single-file build configuration

Important `spec` file settings:

```
console=False  # No console window (GUI only)
upx=True       # Compress binaries
icon='icon.ico'  # Application icon (if available)
```

Testing the Build

```
# After building, test the executable
cd dist
"YouTube Shorts Creator.exe"  # Windows
./YouTube\ Shorts\ Creator    # Linux
```

Code Style & Conventions

Python Style

- Follow PEP 8
- Use type hints where appropriate
- Document all public methods with docstrings
- Keep methods focused (single responsibility)

GUI Conventions

- Prefix private methods with `_`
- Use descriptive variable names
- Group related widgets in frames
- Use ttk widgets over tk widgets (modern look)

Error Handling

- Always catch specific exceptions
- Provide user-friendly error messages
- Log technical details to console
- Never let the GUI crash silently

Common Development Tasks

Adding a New Feature

1. **Update Core Logic** (shorts_creator_core.py)
   `python`
   `def new_feature(self):`
   `"""Your feature implementation"""`
   `pass`

2. **Add GUI Controls** (shorts_creator_gui.py)
   ```python
   # Add widget in _create_widgets()
   self.new_widget = ttk.Button(...)

# Add handler method
def _handle_new_feature(self):
"""Handle new feature"""
pass
```

1. **Update Build Config**
   - Add any new dependencies to requirements.txt
   - Update hiddenimports in build.spec if needed

2. **Test**
   - Test with Python directly: `python shorts_creator_gui.py`
   - Build and test executable
   - Test on clean Windows machine

Debugging

Development Mode

Run directly with Python to see full error messages:

```
python shorts_creator_gui.py
```

Enable Console in Executable

Temporary change in build.spec:

```
console=True  # Shows console window with debug output
```

Rebuild with this change to see stdout/stderr.

Common Issues

Issue: ImportError after building
- **Fix**: Add missing module to `hiddenimports` in build.spec

Issue: Whisper model not found
- **Fix**: Ensure whisper data files are collected in build.spec

Issue: FFmpeg not working
- **Fix**: Ensure imageio-ffmpeg is in requirements.txt

Modifying the UI

All UI code is in `shorts_creator_gui.py`:

```
def _create_widgets(self):
    # main_frame: Main container
    # files_frame: File selection section
    # caption_frame: Caption settings section
    # output_frame: Output location section
    # progress_frame: Progress bar and status
    # console_frame: Console output
    # button_frame: Action buttons
```

Use ttk widgets for modern look:
- `ttk.Frame`: Container
- `ttk.Label`: Text label
- `ttk.Button`: Clickable button
- `ttk.Entry`: Text input
- `ttk.Combobox`: Dropdown menu
- `ttk.Radiobutton`: Radio button
- `ttk.LabelFrame`: Labeled container

Adding Caption Styles

Modify caption appearance in `shorts_creator_core.py`:

```
class ShortsCreator:
    # Current styling
    CAPTION_BG_COLOR = (255, 215, 0)  # Yellow
    CAPTION_TEXT_COLOR = 'black'
    CAPTION_FONT = 'Arial-Bold'
    CAPTION_FONT_SIZE = 50

    # Add new styles here
```

Testing

Manual Testing Checklist

Before releasing:

- [ ] Test with various video formats (MP4, AVI, MOV)
- [ ] Test with different video durations (10s, 30s, 60s)
- [ ] Test all three caption modes (auto, manual, none)
- [ ] Test all Whisper models (tiny, base, small)
- [ ] Test error cases (missing files, invalid files)
- [ ] Test on clean Windows machine (no Python installed)
- [ ] Test with antivirus enabled
- [ ] Test on low-spec computer (4GB RAM)
- [ ] Test with long file paths
- [ ] Test with special characters in filenames

Automated Testing

Currently, the project doesn't have automated tests, but you could add:

```python
# test_core.py
import unittest
from shorts_creator_core import ShortsCreator

class TestShortsCreator(unittest.TestCase):
    def test_video_loading(self):
        # Test video file loading
        pass

    def test_caption_generation(self):
        # Test caption creation
        pass
```

Performance Optimization

Bottlenecks

1. **Whisper Transcription**: Slowest part (2-5 minutes)
   - Use smaller models (tiny/base) for speed
   - Consider caching transcriptions

2. **Video Encoding**: Second slowest (1-3 minutes)
   - Already optimized with preset='medium'
   - Could add quality presets in GUI

3. **Video Loading**: Fast (<10 seconds)

Memory Management

- Videos are processed in-memory
- Large videos (>1080p) are resized, reducing memory
- Clips are closed after processing

Future Optimizations

- Add progress percentage (requires moviepy callbacks)
- Parallel processing for batch creation
- GPU acceleration for video encoding
- Cache Whisper models to avoid re-downloading

Security Considerations

Input Validation

Always validate:
- File paths exist
- File formats are supported
- File sizes are reasonable
- User has write permission for output location

Dependency Security

- Pin versions in requirements.txt
- Regularly update dependencies for security patches
- Use `pip-audit` to check for vulnerabilities

Windows Security

- Code-sign the executable (prevent security warnings)
- Create proper installer with digital signature
- Submit to Windows SmartScreen for reputation

Distribution

Creating a Release

1. **Version Update**
   - Update version in README.md
   - Update version in GUI title

2. **Build**
   `bash`
   `build_windows.bat`

3. **Test**
   - Test on clean Windows 10 machine
   - Test on Windows 11 machine

4. **Package**
   `bash`
   `# Create ZIP with executable`
   `cd dist`
   `7z a -r "YouTube-Shorts-Creator-v2.0-Windows.zip" "YouTube Shorts Creator"`

5. **Upload**
   - Create GitHub release
   - Upload ZIP file
   - Write release notes

Installer Creation (Optional)

Use Inno Setup to create a proper installer:

```
[Setup]
AppName=YouTube Shorts Creator
AppVersion=2.0
DefaultDirName={pf}\YouTube Shorts Creator
DefaultGroupName=YouTube Shorts Creator
OutputDir=installer
OutputBaseFilename=YouTubeShortsCreatorSetup

[Files]
Source: "dist\YouTube Shorts Creator\*"; DestDir: "{app}"; Flags: ignoreversion re-
cursesubdirs

[Icons]
Name: "{group}\YouTube Shorts Creator"; Filename: "{app}\YouTube Shorts Creator.exe"
```

Contributing Guidelines

Pull Request Process

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test thoroughly
5. Update documentation
6. Submit pull request with description

Code Review Checklist

- [ ] Code follows PEP 8
- [ ] All methods have docstrings
- [ ] Error handling is comprehensive
- [ ] No hardcoded paths or values
- [ ] GUI remains responsive during processing
- [ ] Console output is user-friendly
- [ ] Changes are backwards compatible

FAQ for Developers

Q: Why separate core and GUI?
A: Separation of concerns. Core logic can be used in CLI, API, or other interfaces.

Q: Why not use threading.Lock?
A: Only one video is processed at a time (enforced by button states), so locks aren't needed.

Q: Can I use asyncio instead of threading?
A: Yes, but threading is simpler for this use case. MoviePy is synchronous, so async provides no benefit.

Q: How do I add more Whisper languages?
A: Modify `_transcribe_audio()` in shorts_creator_core.py to make language configurable:

```
result = model.transcribe(str(video_path), language=self.language)
```

Q: Can I add video filters/effects?
A: Yes! MoviePy supports many effects. Import from `moviepy.video.fx` and apply to clips.

Resources

- MoviePy Documentation (https://zulko.github.io/moviepy/)
- Whisper Documentation (https://github.com/openai/whisper)
- Tkinter Documentation (https://docs.python.org/3/library/tkinter.html)
- PyInstaller Documentation (https://pyinstaller.org/en/stable/)

Contact

For technical questions or collaboration:
- GitHub Issues: Project Issues (https://github.com/yourusername/youtube-shorts-creator/issues)
- Email: developer@example.com

Happy coding! 🚀