Recording Manager Architecture

Architecture Diagrams

Document Version: 1.0 **Last Updated:** 2025-10-26 **Status:** Implemented - Phase 5 WARNING: **Videos NOT Encrypted** (Encryption planned Phase 6) **Priority:** HIGH - Required for treatment documentation

---

> **WARNING: WARNING - Videos Stored Unencrypted:**
>
> **Current Implementation:** Video recordings are stored as **plaintext MP4 files** without encryption.
>
> - [FAILED] Videos are **NOT encrypted** (HIPAA vulnerability)
> - [FAILED] PHI/PII visible in video (patient faces, treatment areas)
> - [FAILED] DO NOT use for clinical trials (encryption required)
>
> **Future Implementation (Phase 6):** AES-256-GCM encryption (see section below)

Table of Contents

Overview

Purpose

TOSCA records complete video of every treatment session for documentation, quality assurance, and regulatory compliance. This document describes the recording architecture and integration with the session management system.

Recording Objectives:

1. **Documentation:** Visual record of treatment for patient file
2. **Quality Assurance:** Post-treatment review by clinician
3. **Training:** Educational resource for new operators
4. **Compliance:** FDA requires complete treatment records
5. **Legal Protection:** Evidence in case of adverse events

Recording Architecture

System Overview

```
           CameraController
     (src/hardware/camera_controller.py)


              start_recording(output_path, fps)

                       ▼

              cv2.VideoWriter
      (OpenCV Video Encoding)

  - Codec: MP4V (H.264 baseline)
  - Container: MP4
  - Frame rate: 30 FPS (default)
  - Resolution: Camera native (typically 1920x1200)
  - Color: BGR 8-bit


              Frames written continuously

                       ▼

           Video File (MP4)
    data/sessions/<session_id>/recording.mp4

  - Typical size: 500 MB - 2 GB per session
  - Duration: 10-60 minutes
  - Linked to session in database
```

Key Components

CameraController (`src/hardware/camera_controller.py`) - Manages VideoWriter lifecycle - Handles frame capture and encoding - Emits signals for recording status

Session (`src/core/session.py`) - Stores video file path (`video_file_path` field) - Links recording to patient/operator - Tracks session metadata

VideoWriter (`cv2.VideoWriter` from OpenCV) - Encodes frames to MP4 - Handles codec and compression - Thread-safe writes

---

Video Format & Codec

Current Implementation (Phase 5)

Container: MP4 (.mp4)

Video Codec: MP4V (MPEG-4 Part 2)

```
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
writer = cv2.VideoWriter(output_path, fourcc, fps, frame_size)
```

Frame Rate: 30 FPS (configurable)

Resolution: Native camera resolution (typically 1920x1200)

Color Format: BGR 8-bit (OpenCV default)

Compression: Lossy (H.264 baseline profile)

File Naming Convention

```
data/sessions/<session_id>/recording.mp4
```

Example:

```
data/sessions/2025-10-26_143015_subject-042/recording.mp4
```

Metadata in Database:

```
-- sessions table
UPDATE sessions
SET video_path = 'data/sessions/2025-10-26_143015_subject-042/recording.mp4',
    video_size_bytes = 1234567890,
    video_duration_s = 1245.5
WHERE id = 42;
```

Session Integration

Recording Workflow

1. Session Start:

```
# Create session
session = Session(
    session_id="2025-10-26_143015_subject-042",
    subject_id="042",
    operator_id="tech-005",
    start_time=datetime.now()
)

# Start recording
output_path = f"data/sessions/{session.session_id}/recording.mp4"
camera.start_recording(output_path, fps=30)

session.video_file_path = output_path
session.start()
```

2. During Treatment:

```
# Frames automatically captured and encoded
# VideoWriter writes to file continuously

# Recording status available via signal
camera.recording_status_changed.connect(update_ui)
```

3. Session End:

```python
# Stop recording
camera.stop_recording()

# Finalize session
session.complete()

# Save to database
db.save_session(session)
```

CameraController API

start_recording()

```python
def start_recording(
    self,
    output_path: Path,
    fps: float = 30.0,
    codec: str = "mp4v"
) -> bool:
    """
    Start video recording to file.

    Args:
        output_path: Path to output MP4 file
        fps: Frame rate (default 30 FPS)
        codec: FourCC codec (default "mp4v" for H.264)

    Returns:
        True if recording started successfully, False otherwise

    Emits:
        recording_started signal
    """
    # Create output directory if needed
    output_path.parent.mkdir(parents=True, exist_ok=True)

    # Initialize VideoWriter
    fourcc = cv2.VideoWriter_fourcc(*codec)
    self.writer = cv2.VideoWriter(
        str(output_path),
        fourcc,
        fps,
        self.frame_size  # (width, height)
    )

    if not self.writer.isOpened():
        logger.error(f"Failed to open VideoWriter: {output_path}")
        return False

    self.recording = True
    self.recording_started.emit(str(output_path))
    logger.info(f"Recording started: {output_path} @ {fps} FPS")
    return True
```

stop_recording()

```python
def stop_recording(self) -> None:
    """
    Stop video recording and finalize file.

    Emits:
        recording_stopped signal
    """
    if not self.recording or not self.writer:
        return

    # Release VideoWriter (finalizes file)
    self.writer.release()
    self.writer = None

    self.recording = False
    self.recording_stopped.emit()
    logger.info("Recording stopped")
```

Frame Capture (Automatic):

```
# In CameraStreamThread.run():
def frame_callback(cam, stream, frame):
    """Called for every camera frame."""
    frame_data = frame.as_numpy_ndarray()

    # If recording, write frame to video
    if self.writer and self.recording:
        self.writer.write(frame_data)  # ← Thread-safe

    # Also emit for live display
    self.frame_ready.emit(frame_data)
```

Future Encryption

Encryption Strategy (Phase 6)

See: `docs/architecture/08_security_architecture.md` for complete details

Summary: - **Algorithm:** AES-256-GCM (authenticated encryption) - **Key Derivation:** PBKDF2-HMAC-SHA256 from master password - **File Format:** `[12-byte nonce][encrypted video][16-byte auth tag]`

Benefits: - [DONE] Protects PHI (patient video is sensitive data) - [DONE] Tamper-evident (authentication tag) - [DONE] HIPAA compliant (encryption at rest)

Implementation:

```
# Phase 6 implementation
def encrypt_video_file(input_path: Path, output_path: Path, key: bytes) -> None:
    """Encrypt video file with AES-256-GCM."""
    from cryptography.hazmat.primitives.ciphers.aead import AESGCM
    import os

    aesgcm = AESGCM(key)  # 32-byte key for AES-256
    nonce = os.urandom(12)  # 96-bit nonce

    with open(input_path, 'rb') as f:
        plaintext = f.read()

    ciphertext = aesgcm.encrypt(nonce, plaintext, None)

    with open(output_path, 'wb') as f:
        f.write(nonce)
        f.write(ciphertext)  # Includes 16-byte auth tag

# Usage during session end
camera.stop_recording()
encrypt_video_file(
    input_path=session.video_file_path,
    output_path=session.video_file_path.with_suffix('.enc'),
    key=derived_video_key
)
os.remove(session.video_file_path)  # Delete plaintext
```

Playback with Decryption

Planned UI (Phase 6):

```
def playback_encrypted_video(encrypted_path: Path, key: bytes) -> None:
    """Decrypt and playback video."""
    from cryptography.hazmat.primitives.ciphers.aead import AESGCM

    # Read encrypted file
    with open(encrypted_path, 'rb') as f:
        nonce = f.read(12)
        ciphertext = f.read()
```

```python
    # Decrypt
    aesgcm = AESGCM(key)
    plaintext = aesgcm.decrypt(nonce, ciphertext, None)

    # Write to temporary file for playback
    temp_path = Path("temp/playback.mp4")
    with open(temp_path, 'wb') as f:
        f.write(plaintext)

    # Play using cv2.VideoCapture or external player
    play_video(temp_path)

    # Delete temporary file
    temp_path.unlink()
```

Testing

Recording Tests

Test 1: Verify Recording Start/Stop

```python
def test_recording_lifecycle():
    """Test complete recording workflow."""
    camera = CameraController()
    output_path = Path("test_output.mp4")

    # Start recording
    result = camera.start_recording(output_path, fps=30)
    assert result is True
    assert camera.recording is True

    # Simulate frames
    time.sleep(2.0)

    # Stop recording
    camera.stop_recording()
    assert camera.recording is False
    assert output_path.exists()

    # Verify file is valid MP4
    cap = cv2.VideoCapture(str(output_path))
    assert cap.isOpened()
    cap.release()
```

Test 2: Verify Frame Count

```python
def test_frame_count_accuracy():
    """Verify all frames are written."""
    camera = CameraController()
    output_path = Path("test_frames.mp4")

    camera.start_recording(output_path, fps=30)
    time.sleep(3.0)  # Record for 3 seconds
    camera.stop_recording()

    # Count frames in video
    cap = cv2.VideoCapture(str(output_path))
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    cap.release()

    # Expect ~90 frames (30 FPS * 3 seconds)
    assert 85 <= frame_count <= 95  # Allow ±5 frames tolerance
```

Performance Considerations

Disk Space

Estimates:

Session Duration	Resolution	Frame Rate	File Size
10 minutes	1920x1200	30 FPS	~500 MB
30 minutes	1920x1200	30 FPS	~1.5 GB
60 minutes	1920x1200	30 FPS	~3.0 GB

Storage Requirements: - Average session: 30 minutes = 1.5 GB - Daily capacity (8 sessions): ~12 GB - Weekly capacity (40 sessions): ~60 GB - Monthly capacity (160 sessions): ~240 GB

Recommendation: Minimum 1 TB storage for 6 months of sessions

CPU Usage

VideoWriter CPU Impact: - Encoding: ~5-10% CPU (H.264 hardware acceleration if available) - Writing: ~1-2% CPU (disk I/O)

Total: <15% CPU overhead during recording (acceptable)

Memory Usage

Frame Buffers: - Frame size: 1920x1200 x 3 bytes (BGR) = 6.9 MB - Typical buffer: 2-3 frames = ~20 MB - VideoWriter buffer: ~10 MB

Total: ~30 MB memory overhead (negligible)

References

OpenCV

- **VideoWriter:** https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
- **Video Codecs:** https://docs.opencv.org/4.x/dd/d9e/classcv_1_1VideoWriter.html

Video Encryption

- **AES-GCM:** NIST SP 800-38D
- **See:** `docs/architecture/08_security_architecture.md`

Document Owner: Software Architect **Last Updated:** 2025-10-26 **Next Review:** Before Phase 6 (encryption implementation) **Status:** Recording implemented - Encryption planned