IPC Contracts: File Operations

Date: September 16, 2025
Service: File Operations
Namespace: `file:*`

Overview

This contract defines the IPC communication interface for file operations between the renderer and main processes. These operations handle file selection, validation, and save location dialogs.

Contract Definitions

file:select - File Selection Dialog

Purpose: Open file selection dialog and return selected video file paths

Request Interface:

```
interface SelectFilesRequest {
  /** Whether to allow multiple file selection */
  multiple?: boolean
  /** File type filters */
  filters?: Array<{
    name: string
    extensions: string[]
  }>
  /** Default directory to open */
  defaultPath?: string
}
```

Response Interface:

```
interface SelectFilesResponse {
  /** Whether user selected files */
  success: boolean
  /** Selected file paths */
  filePaths: string[]
  /** Error message if selection failed */
  error?: string
}
```

Default Behavior:

- Single file selection if `multiple` not specified

- Video file filters applied by default
- Opens last used directory or user's Videos folder

Error Conditions:

- User cancels dialog: `success: false, filePaths: []`
- System dialog error: `success: false, error: "Dialog error message"`
- Invalid path: `success: false, error: "Invalid default path"`

---

file:validate - Video File Validation

Purpose: Validate selected video file and extract metadata using FFmpeg

Request Interface:

```
interface ValidateFileRequest {
  /** Absolute path to video file */
  filePath: string
  /** Whether to extract detailed metadata */
  includeMetadata?: boolean
}
```

Response Interface:

```
interface ValidateFileResponse {
  /** Whether file is valid */
  isValid: boolean
  /** Validation error message */
  error?: string
  /** File metadata if valid */
  metadata?: {
    name: string
    size: number
    format: string
    duration: number
    resolution: { width: number, height: number }
    bitrate: number
    codec: string
    frameRate: number
  }
  /** Additional technical details */
  details?: {
    streams: any[]
    formatInfo: any
  }
}
```

Validation Process:

1. Check file exists and is accessible
2. Verify file extension is supported
3. Use FFprobe to analyze video content
4. Extract metadata and technical information
5. Validate video is not corrupted

Error Conditions:

- File not found: `isValid: false, error: "File not found"`
- Access denied: `isValid: false, error: "Permission denied"`
- Unsupported format: `isValid: false, error: "Unsupported video format"`
- Corrupted file: `isValid: false, error: "File is corrupted or unreadable"`
- FFmpeg error: `isValid: false, error: "FFmpeg analysis failed: {details}"`

---

file:save-location - Save Location Dialog

Purpose: Open save dialog for selecting output file location

Request Interface:

```
interface SaveLocationRequest {
  /** Default file name */
  defaultPath?: string
  /** Suggested output format */
  format?: string
  /** File type filters */
  filters?: Array<{
    name: string
    extensions: string[]
  }>
}
```

Response Interface:

```
interface SaveLocationResponse {
  /** Whether user selected location */
  success: boolean
  /** Selected output path */
  filePath?: string
  /** Error message if selection failed */
  error?: string
}
```

Default Behavior:

- Uses input filename with converted format extension
- Applies video format filters
- Opens last used output directory or user's Videos folder
- Prevents overwriting existing files without confirmation

Error Conditions:

- User cancels dialog: `success: false`
- Write permission denied: `success: false, error: "Cannot write to selected location"`
- Invalid path: `success: false, error: "Invalid save location"`

Implementation Requirements

Main Process Handler Registration

```
// In electron/main.ts
import { FileOperationsHandlers } from './handlers'

// Register handlers during app initialization
const fileHandlers = new FileOperationsHandlers()
```

Service Integration

```
// In electron/handlers/file-operations.handlers.ts
export class FileOperationsHandlers {
  private fileService: FileOperationsService

  constructor() {
    this.fileService = FileOperationsService.getInstance()
    this.registerHandlers()
  }

  private registerHandlers(): void {
    ipcMain.handle('file:select', this.handleFileSelect.bind(this))
    ipcMain.handle('file:validate', this.handleFileValidate.bind(this))
    ipcMain.handle('file:save-location', this.handleSaveLocation.bind(this))
  }
}
```

Error Handling Pattern

All file operations must implement consistent error handling:

1. Try-catch around all operations
2. Log errors with context
3. Return structured error responses

4. Include user-friendly error messages

Security Considerations

- Validate all file paths to prevent directory traversal
- Check file permissions before operations
- Sanitize user input in dialog options
- Limit file access to appropriate directories

Testing Requirements

Contract Tests

- Verify IPC channel registration
- Test request/response interfaces
- Validate error handling scenarios
- Confirm security constraints

Integration Tests

- End-to-end file selection workflow
- File validation with various formats
- Save location selection process
- Error recovery scenarios

Test Data

- Valid video files in supported formats
- Invalid files (corrupted, wrong format)
- Files with permission restrictions
- Non-existent file paths

Contract Status: ☑ COMPLETE - Ready for Implementation