IPC Contracts: Desktop Video Converter

Date: September 16, 2025
Feature: Desktop Video Converter
Status: Phase 1 Design

Overview

This document defines the Inter-Process Communication (IPC) contracts between the Electron main process and renderer process. All communication follows the secure context bridge pattern with typed interfaces.

Security Model

Context Bridge Architecture

```
// preload/index.ts
import { contextBridge, ipcRenderer } from 'electron'

const electronAPI = {
  // File Operations
  selectFile: () => ipcRenderer.invoke('file:select'),
  selectSaveLocation: (defaultPath?: string) => ipcRenderer.invoke('file:save-
location', defaultPath),
  validateFile: (filePath: string) => ipcRenderer.invoke('file:validate',
filePath),

  // Conversion Operations
  startConversion: (jobConfig: ConversionJobConfig) =>
ipcRenderer.invoke('conversion:start', jobConfig),
  cancelConversion: (jobId: string) => ipcRenderer.invoke('conversion:cancel',
jobId),

  // Progress Listening
  onProgress: (callback: (data: ConversionProgress) => void) => {
    ipcRenderer.on('conversion:progress', (_, data) => callback(data))
  },
  removeProgressListener: () => {
    ipcRenderer.removeAllListeners('conversion:progress')
  },

  // Application State
  getPreferences: () => ipcRenderer.invoke('app:get-preferences'),
  setPreferences: (prefs: UserPreferences) => ipcRenderer.invoke('app:set-
preferences', prefs),

  // System Integration
  showInFileManager: (filePath: string) => ipcRenderer.invoke('system:show-in-
```

```
explorer', filePath),
  openExternal: (url: string) => ipcRenderer.invoke('system:open-external',
url),

  // Application Lifecycle
  getAppInfo: () => ipcRenderer.invoke('app:info'),
  quit: () => ipcRenderer.invoke('app:quit')
}

contextBridge.exposeInMainWorld('electronAPI', electronAPI)
```

File Operations Contracts

File Selection

`file:select`

Displays native file selection dialog for video files.

Request:

```
// No parameters
invoke('file:select'): Promise<FileSelectionResult>
```

Response:

```
interface FileSelectionResult {
  success: boolean
  filePaths: string[]          // Selected file paths (empty if cancelled)
  error?: string               // Error message if failed
}
```

Implementation:

```
// main/ipc-handlers/file-handlers.ts
ipcMain.handle('file:select', async (): Promise<FileSelectionResult> => {
  try {
    const result = await dialog.showOpenDialog({
      properties: ['openFile'],
      filters: [
        {
          name: 'Video Files',
          extensions: ['mp4', 'avi', 'mov', 'mkv', 'wmv', 'flv', 'webm', 'm4v',
'mpg', 'mpeg']
        },
        { name: 'All Files', extensions: ['*'] }
```

```
      ],
      title: 'Select Video File to Convert'
    })

    return {
      success: !result.canceled,
      filePaths: result.filePaths
    }
  } catch (error) {
    return {
      success: false,
      filePaths: [],
      error: error.message
    }
  }
})
```

Save Location Selection

`file:save-location`

Displays save dialog for output file location.

Request:

```
invoke('file:save-location', defaultPath?: string): Promise<SaveLocationResult>
```

Response:

```
interface SaveLocationResult {
  success: boolean
  filePath?: string           // Selected save path
  error?: string              // Error message if failed
}
```

File Validation

`file:validate`

Validates video file using FFprobe and extracts metadata.

Request:

```
invoke('file:validate', filePath: string): Promise<FileValidationResult>
```

Response:

```
interface FileValidationResult {
  success: boolean
  isValid: boolean              // Whether file is valid video
  metadata?: VideoMetadata      // Video metadata if valid
  error?: string                // Error message if invalid
  fileInfo: FileInfo           // Basic file information
}

interface FileInfo {
  name: string                  // File name with extension
  basename: string              // File name without extension
  extension: string             // File extension
  size: number                  // File size in bytes
  lastModified: Date           // Last modification time
  path: string                  // Full file path
}
```

Conversion Operations Contracts

Start Conversion

`conversion:start`

Initiates video conversion with specified settings.

Request:

```
invoke('conversion:start', jobConfig: ConversionJobConfig):
Promise<ConversionStartResult>
```

Types:

```
interface ConversionJobConfig {
  // Input/Output
  inputPath: string
  outputPath: string

  // Conversion Settings
  quality: 'high' | 'medium' | 'low'
  format: 'mp4'                 // Only MP4 supported in MVP
  overwriteExisting: boolean

  // Optional Advanced Settings
  customSettings?: {
    videoBitrate?: number
```

```
    audioBitrate?: number
    resolution?: { width: number; height: number }
    frameRate?: number
  }

  // Job Metadata
  priority: 'normal' | 'high'
  notifyOnComplete: boolean
}

interface ConversionStartResult {
  success: boolean
  jobId?: string               // Unique job identifier
  error?: string               // Error message if failed
  estimatedDuration?: number   // Estimated conversion time in seconds
}
```

Cancel Conversion

`conversion:cancel`

Cancels active conversion job.

Request:

```
invoke('conversion:cancel', jobId: string): Promise<ConversionCancelResult>
```

Response:

```
interface ConversionCancelResult {
  success: boolean
  cancelled: boolean           // Whether job was actually cancelled
  error?: string               // Error message if failed
  cleanupComplete: boolean     // Whether temp files were cleaned
}
```

Progress Communication Contracts

Progress Updates

`conversion:progress` **(Event)**

Sent from main to renderer during active conversion.

Event Data:

```typescript
interface ConversionProgress {
  // Job Identification
  jobId: string

  // Overall Progress
  percent: number              // 0-100 completion percentage
  stage: 'starting' | 'analyzing' | 'converting' | 'finalizing'

  // Detailed Metrics
  frames: {
    current: number            // Frames processed
    total: number              // Total frames
  }

  // Performance
  speed: {
    fps: number                // Current processing FPS
    multiplier: number         // Speed multiplier (1.0 = realtime)
  }

  // Time Information
  time: {
    elapsed: number            // Elapsed seconds
    remaining?: number         // Estimated remaining seconds
    timemark: string           // Current position (HH:MM:SS.mmm)
  }

  // File Size
  size: {
    input: number              // Input file size
    output: number             // Current output size
    estimated?: number         // Estimated final size
  }

  // Quality Metrics
  bitrate: {
    current: number            // Current output bitrate
    target: number             // Target bitrate
  }

  // System Resources
  resources: {
    cpu?: number               // CPU usage percentage
    memory?: number            // Memory usage in MB
  }

  // Status
  canCancel: boolean           // Whether job can be cancelled
  warnings: string[]           // Non-fatal warnings
}
```

Conversion Complete

`conversion:complete` **(Event)**

Sent when conversion finishes successfully.

Event Data:

```
interface ConversionComplete {
  jobId: string
  success: true
  outputPath: string
  statistics: ConversionStatistics
  duration: number              // Total conversion time
}

interface ConversionStatistics {
  // File Comparison
  inputSize: number             // Original file size
  outputSize: number            // Final file size
  compressionRatio: number      // Size reduction percentage

  // Quality Metrics
  originalBitrate: number
  finalBitrate: number
  resolutionChanged: boolean
  frameRateChanged: boolean

  // Performance
  averageFps: number            // Average processing speed
  peakMemoryUsage: number       // Peak memory usage

  // Validation
  outputValid: boolean          // Whether output file is valid
  playable: boolean             // Whether output is playable
}
```

Conversion Error

`conversion:error` **(Event)**

Sent when conversion fails.

Event Data:

```
interface ConversionError {
  jobId: string
  success: false
  error: {
```

```
    code: string              // Error classification
    message: string           // User-friendly error message
    details: string           // Technical error details
    suggestions: string[]     // Suggested solutions
    recoverable: boolean      // Whether retry might succeed
  }
  partialOutput?: string      // Partial file if exists
  cleanup: {
    tempFilesRemoved: boolean
    outputFileRemoved: boolean
  }
}
```

Application State Contracts

Get Preferences

`app:get-preferences`

Retrieves current user preferences.

Request:

```
invoke('app:get-preferences'): Promise<UserPreferences>
```

Response:

```
interface UserPreferences {
  // File Handling
  defaultOutputPath: string
  autoOpenOutputFolder: boolean
  overwriteExisting: boolean

  // Conversion Defaults
  defaultQuality: 'high' | 'medium' | 'low'
  notifyOnComplete: boolean
  notifyOnError: boolean

  // Performance
  ffmpegThreads: 'auto' | number
  maxMemoryUsage: number      // MB limit

  // UI Preferences
  theme: 'system' | 'light' | 'dark'
  showAdvancedOptions: boolean
  minimizeToTray: boolean

  // History
```

```
  saveConversionHistory: boolean
  maxHistoryItems: number

  // Privacy
  analyticsEnabled: boolean
  crashReportingEnabled: boolean
}
```

Set Preferences

`app:set-preferences`

Updates user preferences.

Request:

```
invoke('app:set-preferences', preferences: Partial<UserPreferences>):
Promise<PreferencesUpdateResult>
```

Response:

```
interface PreferencesUpdateResult {
  success: boolean
  updated: string[]            // Keys that were updated
  errors: string[]             // Keys that failed validation
  requiresRestart: boolean     // Whether app restart is needed
}
```

System Integration Contracts

Show in File Manager

`system:show-in-explorer`

Opens file location in Windows Explorer.

Request:

```
invoke('system:show-in-explorer', filePath: string):
Promise<SystemActionResult>
```

Response:

```
interface SystemActionResult {
  success: boolean
  error?: string
}
```

Open External URL

`system:open-external`

Opens URL in default browser.

Request:

```
invoke('system:open-external', url: string): Promise<SystemActionResult>
```

Application Lifecycle Contracts

Get Application Info

`app:info`

Retrieves application metadata.

Request:

```
invoke('app:info'): Promise<AppInfo>
```

Response:

```
interface AppInfo {
  name: string                // Application name
  version: string             // Application version
  electronVersion: string     // Electron version
  nodeVersion: string         // Node.js version
  platform: string            // Operating system
  arch: string                // Architecture

  // Paths
  userDataPath: string        // User data directory
  tempPath: string            // Temporary files directory

  // Capabilities
  ffmpegAvailable: boolean    // Whether FFmpeg is available
  ffmpegVersion?: string      // FFmpeg version if available
```

```
  // Performance
  totalMemory: number          // Total system memory
  freeMemory: number           // Available memory
  cpuCount: number             // Number of CPU cores
}
```

Quit Application

`app:quit`

Gracefully shuts down the application.

Request:

```
invoke('app:quit'): Promise<void>
```

Behavior:

- Cancels any active conversions
- Cleans up temporary files
- Saves user preferences and window state
- Closes application

Error Handling Patterns

Standard Error Response

All IPC calls follow consistent error handling:

```
interface IPCError {
  code: string                 // Error code for programmatic handling
  message: string              // User-friendly error message
  details?: string             // Technical details for debugging
  timestamp: Date              // When error occurred
  retryable: boolean           // Whether operation can be retried
}
```

Common Error Codes

- `FILE_NOT_FOUND`: Specified file does not exist
- `FILE_ACCESS_DENIED`: Insufficient permissions
- `INVALID_VIDEO_FORMAT`: File is not a valid video
- `CONVERSION_FAILED`: FFmpeg conversion error
- `INSUFFICIENT_SPACE`: Not enough disk space
- `PROCESS_CANCELLED`: Operation was cancelled by user

- `SYSTEM_ERROR`: System-level error occurred

Type Definitions Export

```
// shared/types/ipc-contracts.ts
export {
  // File Operations
  FileSelectionResult,
  SaveLocationResult,
  FileValidationResult,
  FileInfo,

  // Conversion
  ConversionJobConfig,
  ConversionStartResult,
  ConversionCancelResult,
  ConversionProgress,
  ConversionComplete,
  ConversionError,
  ConversionStatistics,

  // Application
  UserPreferences,
  PreferencesUpdateResult,
  AppInfo,

  // System
  SystemActionResult,

  // Common
  IPCError
}
```

This IPC contract specification ensures type-safe, secure communication between Electron processes while providing a clean API for the React renderer to interact with system capabilities.