RIP-App Software Design Document

# 1. Project Overview

## 1.1 Project Name:

RIP Application

## **1.2 Project Description:**

The RIP Application is a Cross Platform (Linux & Android) Qt-based job manager and image preprocessor for print workflows. It enables technicians to configure print settings, edit images, generate PRN files, and communicate with network-attached printers using CUPS. It is designed for production print environments that require flexible job configuration, image transformation, and robust output generation pipelines.

## 1.3 Goals and Objectives:

* Provide a simple and robust interface for creating and managing print jobs
* Enable non-destructive image editing and transformation tools
* Support PRN generation for both live CUPS printers and proprietary output for Nocai printers
* Maintain cross-platform compatibility and local-only data processing
* Enable ICC color profile transformations using LittleCMS
* Support FM stochastic screening via blue noise dithering
* Ensure accurate dot sizing and compensation for 2BPP PRN generation

# 2. Functional Requirements

## 2.1 User Features

* **Print Job Management:** Users can create, edit, remove, and save print jobs.
* **Image Editing:** Users can load and modify images via tools like crop, rotate, flip, resize, adjust brightness/contrast, convert color space, and draw shapes or text.
* **Metadata Inspection:** The app extracts and displays image dimensions, channels, and inferred color profiles.
* **PRN Output**: Users can generate PRN files using custom logic for Nocai printers or send jobs to networked printers via CUPS.
* **Printer Setup**: Users can detect and print directly to network-attached printers via CUPS

## 2.2 Administrative Features

* **File Persistence:** Print jobs can be saved and reloaded as JSON with embedded image data.
* **Printer Setup:** Can choose between proprietary Nocai Printers or network printers.
* **Printer Discovery**: Network printers are detected using CUPS.
* **Option Validation**: System queries supported printer options using modern CUPS APIs.

# 3. System Architecture

## 3.1 High-Level Architecture

The backend is written in C++ using Qt 6 and exposes functionality to a QML-based frontend. ImageMagick handles nearly all image processing. LittleCMS is used for ICC color transformations. A custom backend handles dithering, dot classification, and PRN generation.

## 3.2 Technology Stack

* Framework: Qt 6 (C++/QML)
* Image Processing: ImageMagick (Magick++)
* Printing: CUPS and libcups
* File I/O: Qt File and JSON APIs
* Rasterization:Image Magick (Magick++)

## 3.3 Dependencies

* Stb\_image.h (for raw image validation)
* Libcups, libcupsfilters (for printer I/O)
* Magick++ (ImageMagick C++ API)
* Qt Core, GUI, QML modules

## 3.2 Component Modules

**Core Components:**

* PrintJobModel: Central store for all print job information.
* PrintJobOutput: Backend for CUPS printing and network printer communication.
* PrintJobNocai: Backend for 2BPP PRN generation with dot compensation for Nocai printers.
* ImageEditor: Backend image processor using ImageMagick (Magick++).
* MetadataInspector: Utility class for extracting metadata from images, PDFs, and SVGs.

**Front-end UI:**

* QML-based views: Main.qml, JobListView.qml, JobDetailsView.qml, ImageEditorView.qml, PrinterSetupView.qml, ImpositionView.qml

**Communication:**

* All C++ backend classes are exposed to QML via engine.rootContext()->setContextProperty().

# 4. Technical Specifications

## 4.1 PrintJobModel

* Stores list of all print jobs
* Each job includes:
  + imagePath
  + paperSize (enum mapped to QSize)
  + resolution, offset, colorProfile, varnishLayer, whiteInk settings
  + Metadata
* Exposes: add, remove, update, select, and save job features

## 4.2 PrintJobOutput

* Interacts with CUPS to:
  + Detect and list available network printers
  + Register virtual printers
  + Generate PRN files using cupsCreateJob, cupsStartDocument, cupsFinishDocument
* Supports:
  + Real or virtual printers via PPD
  + Custom job attributes

## 4.3 PrintJobNocai

* Creates a PRN compatible with proprietary Nocai Printers
  + Converts CMYK images to 2BPP format with
  + Stochastic blue noise dithering via custom algorithm
  + Dot size classification using 4x4 neighborhood inspection
  + ICC profile transformation via LittleCMS
  + PRN file output with custom headers for Nocai API

## 4.3 ImageEditor

* Uses Magick++ to support:
  + Crop, flip, rotate
  + Brightness and contrast adjustment
  + Color space conversion (RGB, CMYK, grayscale)
  + Drawing shapes and text
  + Temporary file logic with discard/save management
* Exposed directly to QML

## 4.4 MetadataInspector

* Uses stb\_image, poppler, rsvg to extract:
  + DPI, dimensions, color space, ICC profile
  + SVG viewbox & metadata
  + PDF title, author, subject, etc.
* Cached per-file for re-use

# 5. Backend Tools and Algorithms

## 5.1 Image Processing

* All editing operations done via ImageMagick (Magick++)
* High-quality transforms and composite operations

## 5.2 Color Profile Handling

* Uses LittleCMS (lcms2) for ICC conversions
* Supports CMYK, grayscale, RGB, indexed, and custom ICC profiles

## 5.3 FM Stochastic Screening

* Custom random blue noise masks (64x64, 128x128, etc.)
* Per-channel rotation and offset to reduce artifacts
* Applied via ImageMagick -fx logic or threshold blending

## 5.4 Dot Compensation and 2BPP Packing

* Blue noise threshold defines ink-on regions
* Neighborhood logic (4x4) promotes dot size for tonal preservation
* Final format packs 4 pixels per byte (2BPP)
* Output includes Nocai-specific headers and line encoding

# 6. Deployment and Maintenance

## 6.1 Deployment

* **Build system:** qmake or CMake for cross-platform builds
* **Linux:**
  + Requires CUPS development libraries
  + Deployable as AppImage or .deb package
* **Android:**
  + Uses static CUPS build
  + Deploys via Qt for Android packaging
  + All dependencies bundled (Magick++, stb\_image, poppler, librsvg, lcms2)

## 6.2 Maintenance

* Regular updates to Qt libraries, ImageMagick, and LittleCMS
* Monitor deprecated CUPS API usage and maintain compatibility
* Isolate platform-specific code to simplify future portability and maintenance

# 7. Security Considerations

* All files are processed locally, no external data transmission
* Temporary files are deleted after use and not cached unless explicitly saved
* Input file validation and sandboxing for external formats
* Access to printers is performed locally via CUPS.
* Print job data is stored as local JSON files with optional embedded base64 image data.
* Optional checksum verification for job files and image content
* No personal user data is stored; job state exists only in memory or user-saved JSON
* App does not expose external ports

# 8. Performance and Scalability

* **Memory:** Designed for Linux and Android usage, performs well with moderate-sized raster images.
* **Scaling:** Not intended for web or cloud deployment.
* **CPU Acceleration:** ImageMagick utilizes optimized CPU-bound operations.
* **Startup Time:** Under 2 seconds on modern hardware.
* **Job Limit:** Currently unconstrained, limited by memory and file system throughput.

# 9. Testing Strategy

* **Manual Testing:** Performed for all modules across supported formats.
* **File Validation:** Supported image formats are checked via stb\_image and metadata extraction.
* **Editor Consistency:** ImageEditor methods verified for visual and dimensional correctness.
* **PRN Generation:** Simulation of printer output via PRN inspection
* **Color Accuracy:** Confirmed ICC and color space conversions via LittleCMS