PNG to .preserve Conversion Toolchain

Complete Documentation of the Conversion Process

From Raster Image to Editable Design Document

DesignLibre Technical Documentation

January 3, 2026

Contents

1 Executive Summary

This document describes the complete toolchain used to convert a PNG raster image into a DesignLibre `.preserve` file format. The process involves visual analysis, structured data extraction, hierarchical node construction, and archive packaging.

1.1 Key Characteristics

- **Input**: PNG raster image (e.g., `presets-screen.png`)
- **Output**: `.preserve` ZIP archive containing JSON files
- **Processing**: Manual visual analysis + structured JSON generation
- **Tools Used**: Claude AI vision, file system operations, ZIP compression

2 Toolchain Overview

2.1 High-Level Pipeline

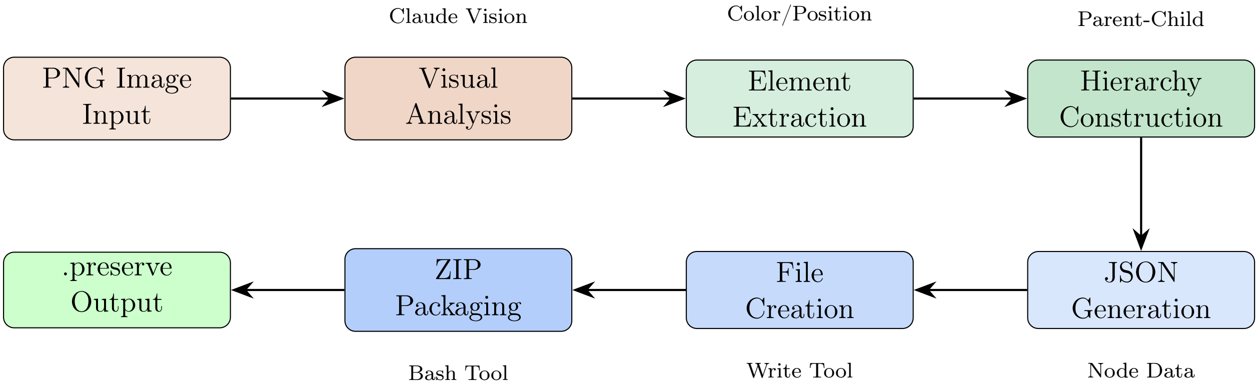

Figure 1: PNG to .preserve Conversion Pipeline

2.2 Tool Calls Summary

Step	Tool	Purpose	Output
1	Read (image)	Load PNG for visual analysis	Image data
2	Glob/Grep	Explore .preserve format spec	Format knowledge
3	Read	Study existing node types	Node structure
4	Write	Create directory structure	Folders
5	Write	Generate JSON files	JSON documents
6	Bash (zip)	Package into archive	.preserve file

Table 1: Tool Invocations in Conversion Process

3 Stage 1: Visual Analysis

3.1 Tool Call: Read (Image File)

The first step is loading the PNG image for visual analysis. Claude's multimodal capabilities allow direct interpretation of raster images.

```
Tool: Read
Parameters: {
  "file_path": "/path/to/presets-screen.png"
}
```

Tool Invocation

3.2 Visual Processing

When Claude reads an image file, the following analysis occurs:

1. **Global Layout Recognition**
   - Device frame identification (iPhone 16 Pro: 393×852)
   - Content area boundaries
   - Grid/alignment detection
2. **Element Identification**
   - Rectangular regions (frames, cards, buttons)
   - Text elements (labels, headings, body text)
   - Icons and decorative elements
   - Background colors and fills
3. **Color Extraction**
   - Dominant colors (background: `#000000`)
   - Accent colors (green: `#618538`)
   - Text colors (white: `#FFFFFF`, gray: `#8E8E93`)
   - UI element colors (cards: `#1A1A1A`)
4. **Spatial Relationships**
   - Nesting hierarchy (which elements contain others)
   - Relative positioning (x, y offsets from parent)
   - Dimensions (width, height of each element)

3.3 Data Extraction Example

From visual analysis of the "Presets" screen:

```
{
  "element": "Search Bar",
  "visual_observations": {
    "position": "top of content area, below status bar",
    "approximate_y": 59,
    "width": "full width with padding",
    "height": "approximately 36px",
    "background_color": "dark gray (#1C1C1E)",
    "corner_radius": "10px (pill shape)",
    "contains": [
      { "type": "icon", "content": "magnifying glass", "color": "#8E8E93" },
      { "type": "text", "content": "Search presets", "color": "#8E8E93" }
    ]
  }
}
```

Extracted Element Data

4 Stage 2: Format Specification Research

4.1 Tool Calls: Glob and Grep

Before generating the .preserve file, the codebase is explored to understand the exact format specification.

```
# Find existing .preserve handling code
Tool: Glob
Parameters: { "pattern": "**/*preserve*" }

# Search for node type definitions
Tool: Grep
Parameters: {
  "pattern": "PreserveNode",
  "path": "src/"
}

# Find archive structure code
Tool: Grep
Parameters: {
  "pattern": "readPreserveArchive",
  "output_mode": "content"
}
```

Format Discovery

4.2 Tool Calls: Read (Source Files)

Key source files are read to understand the data structures:

```
# Node type definitions
Tool: Read
Parameters: { "file_path": "src/preserve/types.ts" }

# Archive structure
Tool: Read
Parameters: { "file_path": "src/preserve/preserve-archive.ts" }

# Node converter (preserve -> scene graph)
Tool: Read
Parameters: { "file_path": "src/preserve/node-converter.ts" }

# Factory functions (how nodes are created)
Tool: Read
Parameters: { "file_path": "src/scene/nodes/factory.ts" }
```

Source File Analysis

4.3 Discovered Format Specification

From the source code analysis, the .preserve format was determined:

```
archive.preserve (ZIP file)
|-- mimetype                        # "application/vnd.designlibre.preserve
   "
|-- document.json                   # Document metadata
|-- pages/
|    |-- page-{id}.json             # Page content with node tree
|-- tokens/
|    |-- colors.json                # Design tokens
|    |-- typography.json
|-- components/
|    |-- component-{id}.json        # Reusable components
|-- assets/
|    |-- manifest.json              # Asset references
|-- prototypes/
|    |-- flows.json                 # Prototype interactions
|-- history/
|    |-- changelog.json             # Version history
|-- META-INF/
     |-- container.xml              # Archive metadata
```

.preserve Archive Structure

5 Stage 3: Hierarchy Construction

5.1 Parent-Child Relationship Mapping

Based on visual analysis, a hierarchical structure is constructed:

```
PAGE: Presets Screen
  |
  +-- FRAME: iPhone 16 Pro (393x852)
        |
        +-- FRAME: Status Bar
        |     +-- TEXT: Time
        |     +-- FRAME: Signal Icons
        |
        +-- FRAME: Content Area
              |
              +-- FRAME: Header Row
              |     +-- TEXT: "Presets"
              |     +-- FRAME: Icons Container
              |
              +-- FRAME: Search Bar
              |     +-- TEXT: "Search presets"
              |
              +-- FRAME: Tab Bar
              |     +-- FRAME: Tab (For You)
              |     +-- FRAME: Tab (Scenes)
              |     +-- FRAME: Tab (Categories)
              |
              +-- FRAME: Preset Card
              |     +-- TEXT: Title
              |     +-- FRAME: Tags Row
              |     +-- FRAME: Play Button
              |
              +-- FRAME: Bottom Navigation
                    +-- FRAME: Nav Item (Presets)
                    +-- FRAME: Nav Item (Timer)
                    +-- FRAME: Nav Item (Settings)
```

Node Hierarchy

5.2 Node ID Generation

Each node requires a unique identifier:

```
// Pattern used for node IDs
const generateNodeId = (descriptor: string): string => {
  // Human-readable prefix + uniqueness
  return ‘${descriptor}-${Date.now().toString(36)}‘;
};

// Examples:
// "iphone-frame"
// "content-area"
// "search-bar"
// "preset-card-1"
```

ID Generation Strategy

6 Stage 4: JSON Generation

6.1 PreserveNode Structure

Each visual element is converted to a PreserveNode JSON object:

```
interface PreserveNode {
  id: string;
  type: ’FRAME’ | ’TEXT’ | ’VECTOR’ | ’GROUP’ | ’IMAGE’;
  name: string;

  transform: {
    x: number;       // Position relative to parent
    y: number;
    width: number;
    height: number;
    rotation: number;
  };

  appearance?: {
    fills?: PreservePaint[];
    strokes?: PreservePaint[];
    strokeWeight?: number;
    cornerRadius?: number;
    opacity?: number;
    effects?: PreserveEffect[];
  };

  // Type-specific properties
  characters?: string;        // For TEXT nodes
  textStyle?: TextStyle;      // For TEXT nodes
  clipContent?: boolean;      // For FRAME nodes

  children?: PreserveNode[];  // Nested nodes
}
```

PreserveNode Interface

6.2 Paint (Fill/Stroke) Structure

```
interface PreservePaint {
  type: 'SOLID' | 'GRADIENT_LINEAR' | 'GRADIENT_RADIAL' | 'IMAGE';
  visible: boolean;
  opacity: number;

  // For SOLID
  color?: { r: number; g: number; b: number; a: number };

  // For gradients
  gradientStops?: Array<{
    position: number;
    color: { r: number; g: number; b: number; a: number };
  }>;
}
```

PreservePaint Interface

6.3 Color Conversion

Colors observed in the PNG are converted to normalized RGBA:

```
// Hex to normalized RGBA
function hexToRgba(hex: string): { r: number; g: number; b: number; a:
   number } {
  const result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
  return {
    r: parseInt(result[1], 16) / 255,  // 0-1 range
    g: parseInt(result[2], 16) / 255,
    b: parseInt(result[3], 16) / 255,
    a: 1
  };
}

// Examples:
// #000000 -> { r: 0, g: 0, b: 0, a: 1 }
// #FFFFFF -> { r: 1, g: 1, b: 1, a: 1 }
// #618538 -> { r: 0.38, g: 0.52, b: 0.22, a: 1 }
// #1C1C1E -> { r: 0.11, g: 0.11, b: 0.12, a: 1 }
```

Color Conversion

6.4 Example Node Generation

```
{
  "id": "search-bar",
  "type": "FRAME",
  "name": "Search Bar",
  "transform": {
    "x": 16,
    "y": 59,
    "width": 361,
    "height": 36,
    "rotation": 0
  },
  "appearance": {
    "fills": [{
      "type": "SOLID",
      "color": { "r": 0.11, "g": 0.11, "b": 0.12, "a": 1 },
      "opacity": 1,
      "visible": true
    }],
    "cornerRadius": 10,
    "opacity": 1
  },
  "clipContent": true,
  "children": [
    {
      "id": "search-placeholder",
      "type": "TEXT",
      "name": "Search Placeholder",
      "transform": {
        "x": 36,
        "y": 9,
        "width": 100,
        "height": 18,
        "rotation": 0
      },
      "characters": "Search presets",
      "textStyle": {
        "fontFamily": "SF Pro",
        "fontSize": 15,
        "fontWeight": 400,
        "fills": [{
          "type": "SOLID",
          "color": { "r": 0.56, "g": 0.56, "b": 0.58, "a": 1 }
        }]
      }
    }
  ]
}
```

Generated Search Bar Node

7 Stage 5: File Creation

7.1 Directory Structure Creation

```
# Create base directory
Tool: Bash
Parameters: {
  "command": "mkdir -p presets-design/{pages,tokens,components,assets,prototypes,history,META-INF}"
}
```

Tool Invocations for Directory Creation

7.2 File Generation Sequence

Each file is created using the Write tool:

```
# 1. Mimetype (must be first, uncompressed in ZIP)
Tool: Write
Parameters: {
  "file_path": "presets-design/mimetype",
  "content": "application/vnd.designlibre.preserve"
}

# 2. Document metadata
Tool: Write
Parameters: {
  "file_path": "presets-design/document.json",
  "content": "<document JSON>"
}

# 3. Page content (main design)
Tool: Write
Parameters: {
  "file_path": "presets-design/pages/page-presets-main.json",
  "content": "<page JSON with all nodes>"
}

# 4. Design tokens
Tool: Write
Parameters: {
  "file_path": "presets-design/tokens/colors.json",
  "content": "<color tokens JSON>"
}
```

```

# 5. Additional metadata files
Tool: Write
Parameters: {
  "file_path": "presets-design/META-INF/container.xml",
  "content": "<container XML>"
}

# ... repeat for all required files
```

File Creation Sequence

7.3 Document.json Structure

```
{
  "$schema": "https://designlibre.app/schemas/preserve/1.0/document.json",
  "id": "doc-presets-screen",
  "name": "Presets Screen Design",
  "version": "1.0.0",
  "created": "2026-01-03T12:00:00.000Z",
  "modified": "2026-01-03T12:00:00.000Z",
  "generator": "Claude AI",
  "generatorVersion": "1.0",
  "pages": [
    {
      "id": "page-presets-main",
      "name": "Presets Screen",
      "path": "pages/page-presets-main.json"
    }
  ],
  "settings": {
    "gridSize": 8,
    "snapToGrid": true,
    "showRulers": true
  }
}
```

document.json Content

8 Stage 6: Archive Packaging

8.1 ZIP Creation

The final step packages all files into a ZIP archive:

```
Tool: Bash
Parameters: {
  "command": "cd presets-design && zip -r ../presets-screen.preserve mimetype document.json pages/ tokens/ components/ assets/ prototypes/ history/ META-INF/"
}
```

ZIP Packaging Command

8.2 ZIP Structure Requirements

1. **Mimetype First**: The `mimetype` file must be the first entry in the ZIP archive (uncompressed) for proper MIME type detection
2. **No Compression for Mimetype**: Use `-0` flag for mimetype if needed
3. **Relative Paths**: All paths inside the ZIP are relative to the archive root

```
# Create ZIP with mimetype first (uncompressed)
cd presets-design
zip -0 ../presets-screen.preserve mimetype
zip -r ../presets-screen.preserve . -x mimetype
```

Proper ZIP Creation (with mimetype first)

9 Complete Tool Call Sequence

9.1 Chronological Tool Invocations

#	Tool	Purpose
1	Read	Load PNG image for visual analysis
2	Glob	Find .preserve-related source files
3	Grep	Search for PreserveNode type definitions
4	Read	Study preserve-archive.ts structure
5	Read	Study node-converter.ts for data mapping
6	Read	Study factory.ts for node creation
7	Read	Study types.ts for interfaces
8	Bash	Create directory structure
9	Write	Create mimetype file
10	Write	Create document.json
11	Write	Create pages/page-presets-main.json
12	Write	Create tokens/colors.json
13	Write	Create tokens/typography.json
14	Write	Create components/index.json
15	Write	Create assets/manifest.json
16	Write	Create prototypes/flows.json
17	Write	Create history/changelog.json
18	Write	Create META-INF/container.xml
19	Bash	Package into ZIP archive

Table 2: Complete Tool Call Sequence

10 Data Flow Diagram

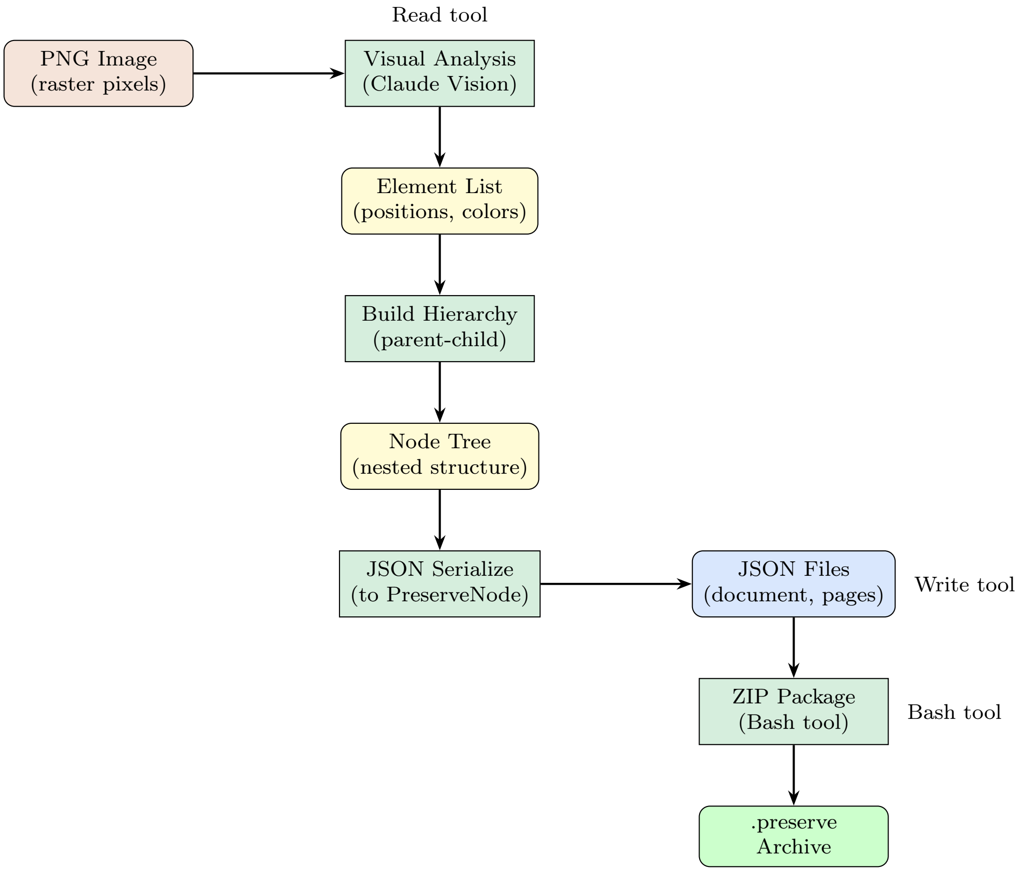
Read tool
PNG Image (raster pixels)
Visual Analysis (Claude Vision)
Element List (positions, colors)
Build Hierarchy (parent-child)
Node Tree (nested structure)
JSON Serialize (to PreserveNode)
JSON Files (document, pages)
Write tool
ZIP Package (Bash tool)
Bash tool
.preserve Archive

Figure 2: Data Flow Through Conversion Pipeline

11 Limitations and Considerations

11.1 Manual Analysis Limitations

1. **Approximate Measurements**: Positions and dimensions are estimated from visual inspection, not pixel-perfect

2. **Font Detection**: Font families are inferred (e.g., "SF Pro" for iOS) rather than extracted

3. **Color Sampling**: Colors are approximated from visual observation

4. **No OCR**: Text content is read visually, not through OCR

5. **Complex Shapes**: Intricate vector paths cannot be perfectly recreated

11.2 Format Fidelity

Property	Accuracy	Method
Position (x, y)	±5px	Visual estimation
Dimensions	±5px	Visual estimation
Colors (solid)	High	Color picker approximation
Corner radius	Medium	Visual estimation
Font size	±2px	Visual estimation
Text content	High	Manual transcription
Hierarchy	High	Logical grouping

Table 3: Property Extraction Accuracy

11.3 Potential Improvements

For automated PNG import, the following enhancements could be implemented:

1. **Computer Vision**: Use edge detection and segmentation for precise boundaries
2. **OCR Integration**: Extract text content programmatically
3. **Color Extraction**: Sample exact pixel colors from the image
4. **AI Segmentation**: Use Claude Vision API with structured output for element detection
5. **Template Matching**: Recognize common UI patterns (buttons, inputs, cards)

12 Conclusion

The PNG to .preserve conversion process involves six distinct stages:

1. **Visual Analysis**: Claude's multimodal capabilities interpret the raster image
2. **Format Research**: Source code exploration reveals the .preserve specification
3. **Hierarchy Construction**: Visual elements are organized into a parent-child tree
4. **JSON Generation**: Each element becomes a PreserveNode with transform and appearance
5. **File Creation**: JSON documents are written to the required directory structure
6. **Archive Packaging**: Files are compressed into a ZIP with .preserve extension

The toolchain leverages Claude's ability to:

- Read and interpret image files visually
- Search and understand source code for format specifications

- Generate structured JSON data from visual observations
- Execute file system operations through tool calls

This process transforms an opaque raster image into an editable, hierarchical design document that can be opened and modified in DesignLibre.