COMPLETE N8N WORKFLOW SYSTEM - FULL IMPLEMENTATION GUIDE

Master Agent + Sub-Workflows with Full Code (v1.119.2)

TABLE OF CONTENTS

ARCHITECTURE OVERVIEW

System Flow

```
┌─────────────────────────────────────────────────────────────┐
│                        INPUTS                               │
├─────────────────────────────────────────────────────────────┤
│  • Webhook (POST /construction-ai)                          │
│  • Telegram Bot (/extract, /convert, etc.)                  │
│  • Direct API Calls                                         │
└─────────────────────────────────────────────────────────────┘
                       │
              ┌────────▼────────┐
              │  00_MASTER_AGENT │
              │  (Main Router)   │
              ├─────────────────┤
              │ ✓ Normalize Input     │
              │ ✓ Validate Request    │
              │ ✓ Route by Type       │
              │ ✓ Execute Sub-Workflow │
              │ ✓ Format Response     │
              │ ✓ Handle Errors       │
```

```
          └───────────┬────────────┘
                    │
      ┌──────────────┼─────────────┬──────────────┐
      │             │             │              │
   ┌───▼───┐    ┌───▼───┐    ┌────▼────┐    ┌────────▼───┐
   │Extract│    │Conver│    │Classify│    │Estimate │
   │ Data  │    │  CAD │    │Material│    │ Cost     │
   └───┬────┘    └───┬───┘    └────┬────┘    └──────┬─────┘
       │            │            │              │
       ├─────────────┼─────────────┼───────────────┤
       │            │            │              │
       └─────────────┼─────────────┼───────────────┘
                   │
          ┌─────────▼─────────┐
          │ RESPONSE       │
          │ Formatter      │
          └─────────┬────────┘
                   │
          ┌─────────▼───────────┐
          │ Output Routes    │
          ├──────────────────────┤
          │ Webhook → JSON    │
          │ Telegram → Text   │
          │ Logger → Records │
          └──────────────────────┘
```

Workflow Categories

Master Orchestrator:

- 00_Master_Agent (Main router and coordinator)

Data Processing:

- 01_Convert_CAD_BIM (DWG, IFC, RVT → JSON)
- 02_Data_Extraction_Agent (PDFs, images → text/data)
- 03_Material_Classification (Classify materials)
- 04_Cost_Estimator (Calculate costs)
- 05_Carbon_Calculator (Carbon footprint)
- 06_Quantity_Takeoff (Extract quantities)

Support:

- 07_Error_Handler (Global error management)
- 08_Data_Validator (Quality assurance)
- 09_Telegram_Handler (Telegram-specific logic)

MASTER AGENT - COMPLETE WORKFLOW

Workflow Structure (JSON-Ready)

This is a **complete, copy-paste ready** Master Agent workflow for N8N v1.119.2:

```
{
  "name": "00_Master_Agent",
  "active": true,
  "nodes": [
    {
      "parameters": {
        "path": "construction-ai",
        "responseMode": "lastNode",
        "options": {}
      },
      "name": "Webhook Trigger",
      "type": "n8n-nodes-base.webhook",
      "typeVersion": 1,
      "position": [250, 100]
    },
    {
      "parameters": {
        "botToken": "{{ env('TELEGRAM_BOT_TOKEN') }}",
        "listenOn": "both"
      },
      "name": "Telegram Trigger",
      "type": "n8n-nodes-telegram.telegram",
      "typeVersion": 1,
      "position": [250, 200]
    },
    {
      "parameters": {
        "jsCode": "// Input Normalizer for Webhook + Telegram\nconst nodeType = $node.cu
      },
      "name": "Input Normalizer",
      "type": "n8n-nodes-base.code",
      "typeVersion": 2,
      "position": [450, 150]
    },
    {
      "parameters": {
        "jsCode": "// Input Validation\nconst input = $json;\nconst errors = [];\n\nif (!
      },
      "name": "Validate Input",
      "type": "n8n-nodes-base.code",
      "typeVersion": 2,
      "position": [650, 150]
    },
    {
      "parameters": {
        "conditions": {
          "vmNodeConditions": [
            {
              "key": "isValid",
              "operation": "equals",
              "value": true
            }
          ]
        }
      },
      "name": "Valid Request?",
      "type": "n8n-nodes-base.if",
      "typeVersion": 1,
      "position": [850, 150]
    },
    {
      "parameters": {
        "conditions": {
          "cases": [
            {
              "condition": {
                "index": 0
              },
              "patternProperties": {
                "requestType": "extract_data"
              }
            },
            {
              "condition": {
                "index": 1
              },
              "patternProperties": {
                "requestType": "convert"
              }
            },
            {
              "condition": {
                "index": 2
              },
              "patternProperties": {
                "requestType": "classify"
              }
            },
            {
              "condition": {
                "index": 3
              },
              "patternProperties": {
                "requestType": "cost_estimate"
              }
            },
            {
              "condition": {
                "index": 4
              },
              "patternProperties": {
                "requestType": "carbon_calculate"
              }
            },
            {
              "condition": {
                "index": 5
              },
              "patternProperties": {
                "requestType": "quantity_takeoff"
              }
            }
          ]
        }
      },
      "name": "Route by Type",
      "type": "n8n-nodes-base.switch",
      "typeVersion": 1,
      "position": [1050, 150]
    },
    {
      "parameters": {
        "workflowId": "02_Data_Extraction_Agent"
      },
      "name": "Execute Extract Data",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 50]
    },
    {
      "parameters": {
        "workflowId": "01_Convert_CAD_BIM"
      },
      "name": "Execute Convert",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 150]
    },
    {
      "parameters": {
        "workflowId": "03_Material_Classification"
      },
      "name": "Execute Classify",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 250]
    },
    {
      "parameters": {
        "workflowId": "04_Cost_Estimator"
      },
      "name": "Execute Cost",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 350]
    },
    {
      "parameters": {
        "workflowId": "05_Carbon_Calculator"
      },
      "name": "Execute Carbon",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 450]
    },
    {
      "parameters": {
        "workflowId": "06_Quantity_Takeoff"
      },
      "name": "Execute Quantity",
      "type": "n8n-nodes-base.executeWorkflow",
      "typeVersion": 1,
      "position": [1250, 550]
    },
    {
      "parameters": {
        "jsCode": "// Format Response\nconst input = $json;\nconst timestamp = new Date()
      },
      "name": "Format Response",
      "type": "n8n-nodes-base.code",
      "typeVersion": 2,
      "position": [1450, 250]
    },
    {
      "parameters": {
        "conditions": {
          "vmNodeConditions": [
            {
              "key": "source",
              "operation": "equals",\n              "value": "telegram"
            }
          ]
        }
      },
      "name": "Is Telegram?",
      "type": "n8n-nodes-base.if",
      "typeVersion": 1,
      "position": [1650, 250]
    },
    {
      "parameters": {
        "botToken": "{{ env('TELEGRAM_BOT_TOKEN') }}",
        "method": "sendMessage\",
        "chatId": \"{{ $json.telegram.chatId }}\",\n        \"text": \"{{ $json.message }
```

SUB-WORKFLOW TEMPLATE STRUCTURE

All Sub-Workflows Follow This Pattern

Start Node - Receive & Validate:

```
// START - Receive input from Master Agent
const input = $json;
```

```
// Validate required fields
if (!input.file &amp;&amp; !input.data) {
  return [{
    json: {
      success: false,
      error: "Missing file or data"
    }
  }];
}

// Pass through to processing
return [{ json: input }];
```

End Node - Format Output:

```
// END - Format output for Master Agent
const input = $json;

return [{
  json: {
    success: true,
    data: {
      result: input.result || input,
      metadata: {
        workflow: $workflow.name,
        processedAt: new Date().toISOString(),
        inputSource: input.source || "unknown"
      }
    },
    message: input.message || "Successfully completed"
  }
}];
```

Error Handler:

```
// ERROR HANDLER - Catch all errors
const error = $error;
const input = $json || {};

return [{
  json: {
    success: false,
    error: {
      message: error.message || "Operation failed",
      type: error.type || "UNKNOWN",
      workflow: $workflow.name
    },
    requestId: input.requestId || "unknown"
  }
}];
```

DATA EXTRACTION WORKFLOW

Workflow: 02_Data_Extraction_Agent

```
// ============================================
// NODE 1: Start - Validate Input
// ============================================
const input = $json;

if (!input.file || !input.file.url) {
  return [{
    json: {
      success: false,
      error: "No file URL provided"
    }
  }];
}

return [{ json: input }];

// ============================================
// NODE 2: Fetch File
// ============================================
// Type: HTTP Request Node
// Method: GET
// URL: {{ $json.file.url }}
// Response Format: File

// ============================================
// NODE 3: OCR Processing (Code Node)
// ============================================
const file = $json;
const fileName = file.name || "document";

// Mock OCR extraction (replace with actual OCR API)
const extractedText = await performOCR(file);

return [{
  json: {
    fileName: fileName,
    extractedText: extractedText,
    confidence: 0.87,
    pages: extractedText.split(/\n\n/).length,
    extractionMethod: "OCR + AI"
  }
}];

// ============================================
// NODE 4: AI Processing (Optional - OpenAI/Gemini)
// ============================================
// Type: OpenAI Chat Model or Gemini
// Prompt: "Extract key information from this text: {{ $json.extractedText }}"
// Returns: Structured data

// ============================================
// NODE 5: Data Cleaning (Code Node)
// ============================================
const input = $json;

const cleaned = {
  document: {
    title: input.title || "",
    content: input.content || "",
    author: input.author || "",
    date: input.date || new Date().toISOString(),
    language: input.language || "en"
  },
  extraction: {
    method: input.extractionMethod || "ocr",
    confidence: input.confidence || 0,
    processedAt: new Date().toISOString()
  },
  tables: input.tables || [],
  images: input.images || [],
  metadata: {
    fileName: input.fileName,
    fileSize: input.fileSize,
    pages: input.pages
  }
};

return [{
  json: {
    success: true,
    result: cleaned,
    message: "Data extraction completed"
  }
}];

// ============================================
// NODE 6: End - Format Output
// ============================================
const input = $json;

return [{
  json: {
    success: true,
    data: {
      result: input.result || input,
      metadata: {
        workflow: "02_Data_Extraction_Agent",
        processedAt: new Date().toISOString()
      }
    },
    message: input.message || "Extraction completed"
  }
}];
```

CAD CONVERSION WORKFLOW

Workflow: 01_Convert_CAD_BIM

```
// ============================================
// NODE 1: Start - Validate CAD File
// ============================================
const input = $json;

const validExtensions = [".dwg", ".dxf", ".rvt", ".ifc", ".dgn"];
if (!validExtensions.includes(input.fileExtension.toLowerCase())) {
  return [{
    json: {
      success: false,
      error: `Invalid file type: ${input.fileExtension}`
    }
  }];
}

return [{ json: input }];

// ============================================
// NODE 2: Fetch CAD File
// ============================================
// Type: HTTP Request Node
// Method: GET
// URL: {{ $json.file.url }}
// Response Format: File

// ============================================
// NODE 3: Parse CAD Geometry (Code Node)
// ============================================
const file = $json;

// Mock CAD parsing (replace with actual library like Tetrahedralize)
const geometry = {
  points: [],
  lines: [],
  polylines: [],
  circles: [],
  arcs: [],
  text: [],
  dimensions: []
};

// Process based on file type
const ext = file.extension.toLowerCase();
if (ext === ".dwg") {
  geometry = await parseDWG(file);
} else if (ext === ".ifc") {
  geometry = await parseIFC(file);
} else if (ext === ".rvt") {
  geometry = await parseRevit(file);
}

return [{
  json: {
    geometry: geometry,
    fileType: ext,
    layerCount: geometry.layers?.length || 0
  }
}];

// ============================================
// NODE 4: Extract Layers &amp; Properties (Code Node)
// ============================================
const input = $json;

const layers = {
  count: input.geometry.layers?.length || 0,
  layers: input.geometry.layers || [],
  properties: input.geometry.properties || {}
};

const properties = {
  boundingBox: {
    minX: input.geometry.minX || 0,
    minY: input.geometry.minY || 0,
    minZ: input.geometry.minZ || 0,
    maxX: input.geometry.maxX || 100,
    maxY: input.geometry.maxY || 100,
    maxZ: input.geometry.maxZ || 100
  },
  units: input.geometry.units || "millimeters",
  scale: input.geometry.scale || 1,
  coordinateSystem: input.geometry.coordinateSystem || "cartesian"
};

return [{
  json: {
    layers: layers,
    properties: properties,
    elementCount: {
      points: input.geometry.points?.length || 0,
      lines: input.geometry.lines?.length || 0,
      polylines: input.geometry.polylines?.length || 0
    }
  }
}];

// ============================================
// NODE 5: Convert to GeoJSON (Code Node)
// ============================================
const input = $json;

const geoJSON = {
  type: "FeatureCollection",
  features: []
};

// Convert geometry to GeoJSON features
if (input.geometry.points) {
    input.geometry.points.forEach(pt =&gt; {
      geoJSON.features.push({
        type: "Feature",\n      geometry: {\n        type: "Point\",\n        coordinates:
```