

Project KickOff Preparations CheckList

Source File: generated-documents\planning\Project-KickOff-Preparations-CheckList.md

Generated: 08/07/2025 at 09:42:27

Generated by: Requirements Gathering Agent - PDF Converter

Project KickOff Preparations Checklist

Generated by adpa-enterprise-framework-automation v3.1.6

Category: planning-artifacts

Generated: 2025-07-05T17:07:36.793Z

Description: Checklist for project kickoff preparations, including scope, stakeholders, resources, and readiness.

Project KickOff Preparations Checklist for === PROJECT README ===

Let's brainstorm a completely different and ambitious project: "Self-Charging Electric Vehicles" (SCEV).

This is a fascinating concept that tackles one of the biggest hurdles for electric vehicle adoption. Here's a breakdown of the idea.

Project Idea: The "Perpetual Motion" EV

1. The Elevator Pitch

We are developing a new class of electric vehicles that significantly reduce the need to plug in by harvesting ambient energy from their environment. By integrating advanced solar, kinetic, and thermal energy recovery systems, the vehicle constantly "trickle-charges" itself during driving and even while parked, dramatically extending its effective range and reducing reliance on traditional charging infrastructure.

2. The Problem It Solves

Range Anxiety: The single biggest fear for potential EV buyers. Our system directly counters this by continuously adding miles back to the battery.

Charging Infrastructure Gaps: In many urban and rural areas, reliable public charging is scarce. This project makes EVs viable for a much wider audience.

Grid Strain: A massive influx of EVs will put an enormous strain on the electrical grid. Self-charging vehicles lessen this load by generating a portion of their own power.

Cost and Inconvenience: Reduces the time and money spent at charging stations and the hassle of installing a home charger.

3. Core Technologies to Integrate

This isn't about a single solution, but a holistic system of multiple energy-harvesting technologies managed by a central AI.

1. Advanced Photovoltaic Body Panels:

Concept: Instead of a simple solar roof, the car's entire body—hood, roof, trunk, and even doors—is constructed from a lightweight, durable composite material with integrated, high-efficiency solar cells.

Innovation: Using new perovskite or multi-junction solar cells that are more efficient, flexible, and perform better in low-light conditions than

traditional silicon.

2. Regenerative Suspension System:

Concept: Standard regenerative braking captures energy when slowing down. We'll add a system that captures energy from the vertical movement of the suspension.

Innovation: Each shock absorber is replaced with a linear electromagnetic generator. Every bump, pothole, and body roll during a turn generates electricity by moving magnets through coils, turning wasted kinetic energy into usable power.

3. Thermoelectric Generation (TEG):

Concept: Capture waste heat from various sources and convert it into electricity.

Innovation: TEG modules would be placed on the battery pack, electric motors, and radiator. As these components heat up during operation, the temperature difference is used to generate a steady stream of power.

4. AI-Powered Energy Management Unit (EMU):

Concept: The "brain" of the system. It's not enough to just generate power; it must be managed intelligently.

Innovation: The EMU uses machine learning to:

Predict energy generation: It analyzes weather forecasts (sunlight), GPS route data (hills, rough roads), and driving style to predict how much energy can be harvested.

Optimize energy flow: It decides in real-time whether to send harvested energy directly to the motors for immediate use or to the battery for storage, based on the current state of charge and predicted needs.

Provide user feedback: A dashboard shows the driver in real-time how much energy is being generated from each source (solar, kinetic, thermal).

4. First Few Project Milestones

M1: Component Feasibility & Simulation: Research and benchmark the most promising solar, kinetic, and thermoelectric technologies. Create a detailed digital twin of a standard EV to simulate the potential energy gains under various real-world conditions (e.g., a sunny commute in Arizona vs. a bumpy, overcast day in Seattle).

M2: Prototype Development: Build and lab-test a functional prototype of the three core hardware systems: a car hood made of photovoltaic composite, a single regenerative shock absorber, and a TEG unit for a battery pack.

M3: Test Mule Integration: Retrofit an existing electric vehicle (the "test mule") with the prototype hardware. The goal is not full integration, but to mount the systems and collect real-world performance data.

M4: Energy Management Unit (EMU) v1.0: Develop the initial software and hardware for the EMU. In this phase, it will only need to accurately read data from all the new sensors and log it for analysis. Control logic will come in a later milestone.

This project represents a fundamental shift from thinking of an EV as a device that simply consumes power to one that actively participates in its own energy lifecycle.

=== PROJECT METADATA ===

Name: adpa-enterprise-framework-automation

Description: Modular, standards-compliant Node.js/TypeScript automation framework for enterprise requirements, project, and data management. Provides CLI and API for BABOK v3, PMBOK 7th Edition, and DMBOK 2.0 (in progress). Production-ready Express.js API with TypeSpec architecture. Designed for secure, scalable, and maintainable enterprise automation.

Version: 3.1.6

Dependencies: @azure-rest/ai-inference, @azure/identity, @azure/msal-node, @azure/openai, @google/generative-ai, @microsoft/microsoft-graph-client, axios, bcryptjs, compression, cors, dotenv, express, express-rate-limit, express-validator, express-winston, form-data, glob, helmet, joi, jsonwebtoken, morgan, multer, node-fetch, openai, swagger-ui-express, ts-node, uuid, winston, yargs, zod

Dev Dependencies: @jest/globals, @redocly/cli, @types/bcryptjs, @types/compression, @types/cors, @types/express, @types/glob, @types/jest, @types/jsonwebtoken, @types/morgan, @types/multer, @types/node, @types/node-fetch, @types/swagger-ui-express, @types/uuid, @typespec/compiler, @typespec/http, @typespec/json-schema, @typespec/openapi3, @typespec/rest, ajv, jest, rimraf, ts-jest,

typescript

Available Scripts: build, copy-configs, start, api:start, dev, clean, test, test:providers, test:performance, test:azure, test:github, test:ollama, test:failover, test:unit, prepublishOnly, admin:install, admin:dev, admin:build, admin:start, admin:setup, admin:serve, confluence:init, confluence:test, confluence:oauth2:login, confluence:oauth2:status, confluence:oauth2:debug, confluence:publish, confluence:status, sharepoint:init, sharepoint:test, sharepoint:oauth2:login, sharepoint:oauth2:status, sharepoint:oauth2:debug, sharepoint:publish, sharepoint:status, api:compile, api:watch, api:format, api:lint, api:docs, api:serve-docs, api:demo, api:server, babok:generate, pmbok:generate, dmbok:generate, framework:multi

=== DEMONSTRATION-GUIDE.MD (documentation) ===

Path: ADPA\DEMONSTRATION-GUIDE.md

Relevance Score: 80



ADPA Markdown-to-Word Integration - Complete Demonstration Guide



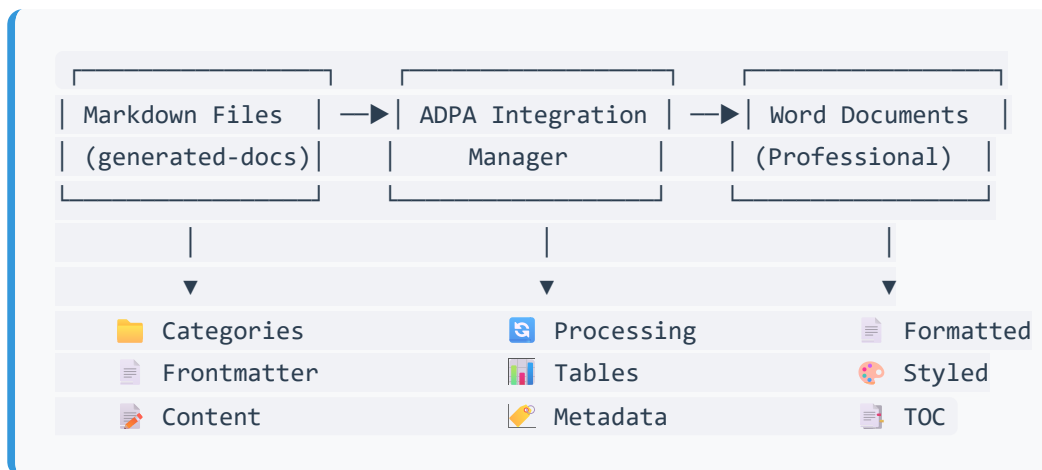
Overview

This guide demonstrates how the ADPA (Automated Documentation Project Assistant) seamlessly converts markdown files from your requirements-gathering workflow into professional Word documents with PMBOK-style formatting.



System Architecture

Document Processing Pipeline






Live Demonstration Workflow

Step 1: Document Discovery

The system automatically scans your `generated-documents/` folder and discovers:

Categories Found:

-  **Project Charter** (1 document)
 - Project Charter: ADPA System
-  **Planning** (4 documents)
 - Work Breakdown Structure
 - Project Management Plan
 - Risk Management Plan
 - Communication Plan
-  **Requirements** (3 documents)
 - Business Requirements Specification
 - Functional Requirements
 - System Requirements

Step 2: Single Document Conversion

Before: Raw Markdown

```
---
title: "Project Charter"
category: "project-charter"
author: "ADPA System"
version: "1.0"
---

# Project Charter: ADPA

## Executive Summary
This Project Charter authorizes the initiation...

## Project Objectives
| Objective | Success Criteria | Timeline |
|-----|-----|-----|
| Automate Documentation | 95% accuracy | Q2 2025 |
```

•

... [truncated]

=== ADOBE-CREDENTIALS-SETUP.MD (primary) ===

Path: ADPA\ADOBE-CREDENTIALS-SETUP.md

Relevance Score: 65

How to Get Your Adobe.io Credentials

Step 1: Access Adobe Developer Console

1. Go to <https://developer.adobe.com/console>
2. Sign in with your Adobe ID (the same one you use for Adobe Creative Cloud)

Step 2: Find or Create Your Project

If you already have a project:

1. Click on your existing project
2. Go to the **Credentials** section

If you need to create a new project:

1. Click **Create new project**
2. Give it a name like "ADPA Document Processing"
3. Click **Create**

Step 3: Add Adobe PDF Services API

1. In your project, click **Add API**
2. Find **Adobe PDF Services API** in the list
3. Click **Next**
4. Choose **Server-to-Server** authentication
5. Click **Save configured API**

Step 4: Get Your Credentials

After adding the API, you'll see your credentials:

Copy these values to your .env file:

```
# From the "Credentials" section in Adobe Developer Console:  
  
ADOBE_CLIENT_ID=your_client_id_from_console  
ADOBE_CLIENT_SECRET=your_client_secret_from_console  
ADOBE_ORGANIZATION_ID=your_org_id_from_console
```

Where to find each value:

- **ADOBE_CLIENT_ID**: Listed as "Client ID" in the credentials section

- **ADOBE_CLIENT_SECRET**: Listed as "Client Secret" (click "Retrieve client secret")
- **ADOBE_ORGANIZATION_ID**: Listed as "Organization ID" at the top of the console

Step 5: Test Your Credentials

Run this test to make sure your credentials work:

```
curl -X POST "https://ims-na1.adobelogin.com/ims/token" \  
-H "Content-Type: application/x-www-form-urlencoded" \  
-d "grant_type=client_credentials&client_id=YOUR_CLIENT_ID&client_se
```

If successful, you'll get back an access token!

Step 6: Update Your .env File

1. Open your `.env` file in the ADPA directory
2. Replace the placeholder values with your actual credentials:

```
ADOBE_CLIENT_ID=abcd1234efgh5678    # Your actual Client ID  
ADOBE_CLIEN  
... [truncated]  
  
=== CLI-REFACTOR-IMPLEMENTATION-GUIDE.MD (documentation) ===  
Path: CLI-REFACTOR-IMPLEMENTATION-GUIDE.md  
Relevance Score: 62  
  
# CLI Refactor Implementation Guide  
  
This guide outlines the recommended steps to refactor the Requirements  
  
---  
  
## 1. Adopt a CLI Framework (Yargs)  
  
**Why:**
```

- Simplifies argument parsing and validation
- Automatically generates help output
- Makes commands and options declarative and discoverable

****How:****

- Install Yargs:

```
``sh
npm install yargs @types/yargs
```

- Refactor `src/cli.ts` to use Yargs for all command and option parsing.
- Example starter:

```
import yargs from 'yargs';
import { hideBin } from 'yargs/helpers';

yargs(hideBin(process.argv))
  .command('generate [key]', 'Generate a document', (yargs) => {
    yargs.positional('key', { type: 'string', describe: 'Document'
  }, (argv) => {
    // Call generate logic
  })
  .option('output', { type: 'string', default: 'generated-document'
  .help()
  .argv;
```

- Remove all manual `process.argv` parsing and helper functions.

Common Issues & Solutions:

- **Duplicate imports:** Remove old import statements when adding Yargs imports
 - **Type errors:** Use proper type assertions for argv values (e.g., `argv.format as 'markdown' | 'json' | 'yaml'`)
 - **Missing modules:** Create placeholder functions or use dynamic imports for non-essential commands during transition
-

2. Increase Modularity (Command Extraction)

Why:

- Easier to maintain and extend
- Each command is independently testable

How:

- Create a `src/commands/` directory.
- Move logic for each major command (e.g., generate, confluence, sharepoint, vcs) into its own file:

```
src/  
  commands/  
    generate.ts  
    confluence.ts  
    sharepoint.ts  
    vcs.ts  
  utils/  
    validation.ts
```

... [truncated]

=== STEP-2-COMPLETION-SUMMARY.MD (other) ===

Path: STEP-2-COMPLETION-SUMMARY.md

Relevance Score: 51

Step 2 Implementation Complete: Increased Modularity (Command Extraction)



What Was Accomplished

Command Modules Created

- `src/commands/confluence.ts` - Confluence integration commands
- `src/commands/sharepoint.ts` - SharePoint integration commands
- `src/commands/vcs.ts` - Version Control System commands
- `src/commands/utils/validation.ts` - Input validation utilities
- `src/commands/utils/common.ts` - Shared command utilities

Command Structure Implemented

```
src/  
  commands/  
    analyze.ts      ✓ (existing)  
    confluence.ts   ✓ (new)  
    generate.ts     ✓ (existing, enhanced)  
    index.ts        ✓ (updated)  
    setup.ts        ✓ (existing)  
    sharepoint.ts   ✓ (new)  
    status.ts       ✓ (existing)  
    validate.ts     ✓ (existing)  
    vcs.ts          ✓ (new)  
    utils/  
      common.ts     ✓ (new)  
      validation.ts  ✓ (new)
```

New CLI Commands Added

Confluence Commands

- `rga confluence init` - Initialize Confluence configuration
- `rga confluence test` - Test Confluence connection
- `rga confluence publish` - Publish documents to Confluence
- `rga confluence status` - Show Confluence integration status
- `rga confluence oauth2 login` - Start OAuth2 authentication
- `rga confluence oauth2 status` - Check OAuth2 authentication status
- `rga confluence oauth2 debug` - Debug OAuth2 authentication

SharePoint Commands

- `rga sharepoint init` - Initialize SharePoint configuration
- `rga sharepoint test` - Test SharePoint connection
- `rga sharepoint publish` - Publish documents to SharePoint
- `rga sharepoint status` - Show SharePoint integration status
- `rga sharepoint oauth2 login` - Start OAuth2 authentication
- `rga sharepoint oauth2 status` - Check OAuth2 authentication status
- `rga sharepoint oauth2 debug` - Debug OAuth2 authentication

VCS Commands

- `rga vcs init` - Initialize Git repository
... [truncated]

PMBOK Planning Artifacts Checklist

Status	Artifact	Quality	Issues
[]	Work Breakdown Structure	-	Missing
[]	WBS Dictionary	-	Missing
[]	Activity List	-	Missing
[]	Activity Duration Estimates	-	Missing
[]	Activity Resource Estimates	-	Missing
[]	Schedule Network Diagram	-	Missing
[]	Milestone List	-	Missing
[]	Schedule Development Input	-	Missing

1. Project Initiation

- ☐ **Confirm Project Sponsor & Product Owner** identified and engaged
 - ☐ **Define and align project vision, objectives, and success criteria** with stakeholders
 - ☐ **Review and finalize project scope** including all deliverables (PMBOK-compliant docs, modular CLI, Azure AI integration)
 - ☐ **Identify all key stakeholders** (PMs, BAs, Developers, Compliance, IT leadership, Vendors) and establish communication plan
 - ☐ **Document roles and responsibilities** for all team members and stakeholders
 - ☐ **Set up project governance framework** aligned with PMBOK standards and organizational policies
-

2. Planning and Requirements

- ☐ **Gather and confirm detailed functional requirements** based on user stories and acceptance criteria, including:
 - Project Charter generation
 - Stakeholder Register
 - Requirements Management Plan
 - Technology Stack Analysis
 - Risk Management Plan
 - WBS and WBS Dictionary
 - Quality Management Plan
 - Compliance Considerations documentation
 - CLI interface with modular generation
- ☐ **Define non-functional requirements:** security, performance, compliance, integration, modularity
- ☐ **Confirm JSON schema definitions** for all generated documents and output formats

- ☐ **Establish AI integration requirements:** Azure AI API usage limits, authentication mechanisms, usage metrics
 - ☐ **Identify regulatory and compliance requirements** with Compliance Officers' input
 - ☐ **Plan for user adoption and training needs** (documentation, tutorials, support)
-

3. Project Setup

- ☐ **Provision Azure resources** needed for AI integration (Azure OpenAI, Identity)
 - ☐ **Establish secure credential management process** (Azure Key Vault, environment variables, secrets scanning)
 - ☐ **Set up development environment:** Node.js/TypeScript, IDEs, version control (Git), package managers
 - ☐ **Configure project repositories** with proper branch policies and access control
 - ☐ **Establish CI/CD pipelines** for build, test, and deployment automation
 - ☐ **Define coding standards and review processes** (TypeScript best practices, linting)
 - ☐ **Set up testing infrastructure:** unit, integration, and schema validation frameworks
-

4. Risk Management

- ☐ **Conduct risk workshop** to validate and refine risk register based on project specifics
- ☐ **Assign risk owners and mitigation strategies** for key risks (AI accuracy, security, compliance, dependencies)
- ☐ **Implement monitoring and alerting** for Azure AI usage, API limits, and system health
- ☐ **Plan for fallback mechanisms and error handling** in case of Azure AI service issues

- ☐ **Define data privacy and governance protocols** for AI data handling and storage
-

5. Communication & Coordination

- ☐ **Establish regular project status meetings** and reporting cadence
 - ☐ **Set up collaboration platforms** (Slack, MS Teams, Confluence, Jira) with appropriate channels and permissions
 - ☐ **Define escalation paths** for issues and decisions
 - ☐ **Coordinate with Compliance Officers and PMO Admins** for ongoing alignment and approvals
 - ☐ **Schedule stakeholder demos and feedback sessions** for iterative validation
-

6. Deliverables & Milestones Definition

- ☐ Define milestone schedule including:
 - Prototype/alpha release (basic doc generation)
 - Beta with AI integration and CLI commands
 - Full feature completion with all document types and modular API
 - QA and UAT completion
 - Final release and deployment
 - ☐ Define deliverable acceptance criteria based on user stories and PMBOK standards
 - ☐ Plan for documentation and training materials delivery
-

7. Security & Compliance

- ☐ Review and approve security policies related to credentials, data handling, and access control

- ☐ Define audit logging and traceability requirements for AI calls and document generation
 - ☐ Ensure compliance with relevant regulations and organizational standards
 - ☐ Plan for regular security reviews and penetration testing if applicable
-

8. Training & Support

- ☐ Assign support and training roles
 - ☐ Develop user guides, CLI usage docs, FAQs, and training sessions
 - ☐ Plan for feedback collection and issue tracking post-launch
-

9. Kickoff Meeting Agenda (Initial)

- ☐ Introductions and role clarifications
 - ☐ Review project objectives, scope, and success metrics
 - ☐ Present project plan, milestones, and deliverables
 - ☐ Discuss risks, challenges, and mitigation strategies
 - ☐ Confirm communication and collaboration tools
 - ☐ Open Q&A and align on next steps
-

Summary

This checklist ensures that the === PROJECT README ===

Let's brainstorm a completely different and ambitious project: "Self-Charging Electric Vehicles" (SCEV).

This is a fascinating concept that tackles one of the biggest hurdles for electric vehicle adoption. Here's a breakdown of the idea.

Project Idea: The "Perpetual Motion" EV

1. The Elevator Pitch

We are developing a new class of electric vehicles that significantly reduce the need to plug in by harvesting ambient energy from their environment. By integrating advanced solar, kinetic, and thermal energy recovery systems, the vehicle constantly "trickle-charges" itself during driving and even while parked, dramatically extending its effective range and reducing reliance on traditional charging infrastructure.

2. The Problem It Solves

Range Anxiety: The single biggest fear for potential EV buyers. Our system directly counters this by continuously adding miles back to the battery.

Charging Infrastructure Gaps: In many urban and rural areas, reliable public charging is scarce. This project makes EVs viable for a much wider audience.

Grid Strain: A massive influx of EVs will put an enormous strain on the electrical grid. Self-charging vehicles lessen this load by generating a portion of their own power.

Cost and Inconvenience: Reduces the time and money spent at charging stations and the hassle of installing a home charger.

3. Core Technologies to Integrate

This isn't about a single solution, but a holistic system of multiple energy-harvesting technologies managed by a central AI.

1. Advanced Photovoltaic Body Panels:

Concept: Instead of a simple solar roof, the car's entire body—hood, roof, trunk, and even doors—is constructed from a lightweight, durable composite material with integrated, high-efficiency solar cells.

Innovation: Using new perovskite or multi-junction solar cells that are more efficient, flexible, and perform better in low-light conditions than traditional silicon.

2. Regenerative Suspension System:

Concept: Standard regenerative braking captures energy when slowing down. We'll add a system that captures energy from the vertical movement of the suspension.

Innovation: Each shock absorber is replaced with a linear electromagnetic generator. Every bump, pothole, and body roll during a turn generates electricity by moving magnets through coils, turning wasted kinetic energy into usable power.

3. Thermoelectric Generation (TEG):

Concept: Capture waste heat from various sources and convert it into electricity.

Innovation: TEG modules would be placed on the battery pack, electric motors, and radiator. As these components heat up during operation, the temperature difference is used to generate a steady stream of power.

4. AI-Powered Energy Management Unit (EMU):

Concept: The "brain" of the system. It's not enough to just generate power; it must be managed intelligently.

Innovation: The EMU uses machine learning to:

Predict energy generation: It analyzes weather forecasts (sunlight), GPS route data (hills, rough roads), and driving style to predict how much energy can be harvested.

Optimize energy flow: It decides in real-time whether to send harvested energy directly to the motors for immediate use or to the battery for storage, based on the current state of charge and predicted needs.

Provide user feedback: A dashboard shows the driver in real-time how much energy is being generated from each source (solar, kinetic, thermal).

4. First Few Project Milestones

M1: Component Feasibility & Simulation: Research and benchmark the most promising solar, kinetic, and thermoelectric technologies. Create a detailed digital twin of a standard EV to simulate the potential energy gains under various real-world conditions (e.g., a sunny commute in Arizona vs. a bumpy, overcast day in Seattle).

M2: Prototype Development: Build and lab-test a functional prototype of the three core hardware systems: a car hood made of photovoltaic composite, a single regenerative shock absorber, and a TEG unit for a battery pack.

M3: Test Mule Integration: Retrofit an existing electric vehicle (the "test mule") with the prototype hardware. The goal is not full integration, but

to mount the systems and collect real-world performance data.

M4: Energy Management Unit (EMU) v1.0: Develop the initial software and hardware for the EMU. In this phase, it will only need to accurately read data from all the new sensors and log it for analysis. Control logic will come in a later milestone.

This project represents a fundamental shift from thinking of an EV as a device that simply consumes power to one that actively participates in its own energy lifecycle.

=== PROJECT METADATA ===

Name: adpa-enterprise-framework-automation

Description: Modular, standards-compliant Node.js/TypeScript automation framework for enterprise requirements, project, and data management. Provides CLI and API for BABOK v3, PMBOK 7th Edition, and DMBOK 2.0 (in progress). Production-ready Express.js API with TypeSpec architecture. Designed for secure, scalable, and maintainable enterprise automation.

Version: 3.1.6

Dependencies: @azure-rest/ai-inference, @azure/identity, @azure/msal-node, @azure/openai, @google/generative-ai, @microsoft/microsoft-graph-client, axios, bcryptjs, compression, cors, dotenv, express, express-rate-limit, express-validator, express-winston, form-data, glob, helmet, joi, jsonwebtoken, morgan, multer, node-fetch, openai, swagger-ui-express, ts-node, uuid, winston, yargs, zod

Dev Dependencies: @jest/globals, @redocly/cli, @types/bcryptjs, @types/compression, @types/cors, @types/express, @types/glob, @types/jest, @types/jsonwebtoken, @types/morgan, @types/multer, @types/node, @types/node-fetch, @types/swagger-ui-express, @types/uuid, @typespec/compiler, @typespec/http, @typespec/json-schema, @typespec/openapi3, @typespec/rest, ajv, jest, rimraf, ts-jest, typescript

Available Scripts: build, copy-configs, start, api:start, dev, clean, test, test:providers, test:performance, test:azure, test:github, test:ollama, test:failover, test:unit, prepublishOnly, admin:install, admin:dev, admin:build, admin:start, admin:setup, admin:serve, confluence:init, confluence:test, confluence:oauth2:login, confluence:oauth2:status,

confluence:oauth2:debug, confluence:publish, confluence:status,
sharepoint:init, sharepoint:test, sharepoint:oauth2:login,
sharepoint:oauth2:status, sharepoint:oauth2:debug, sharepoint:publish,
sharepoint:status, api:compile, api:watch, api:format, api:lint, api:docs,
api:serve-docs, api:demo, api:server, babok:generate, pmbok:generate,
dmbok:generate, framework:multi

=== DEMONSTRATION-GUIDE.MD (documentation) ===

Path: ADPA\DEMONSTRATION-GUIDE.md

Relevance Score: 80

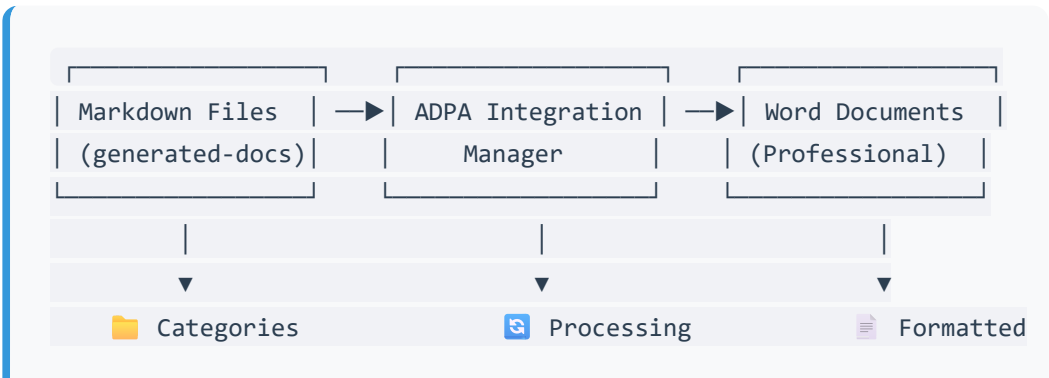
ADPA Markdown-to-Word Integration - Complete Demonstration Guide







Overview

This guide demonstrates how the ADPA (Automated Documentation Project Assistant) seamlessly converts markdown files from your requirements-gathering workflow into professional Word documents with PMBOK-style formatting.

System Architecture

Document Processing Pipeline





 Frontmatter	 Tables	 Styled
 Content	 Metadata	 TOC

Live Demonstration Workflow

Step 1: Document Discovery

The system automatically scans your `generated-documents/` folder and discovers:

Categories Found:

-  **Project Charter** (1 document)
 - Project Charter: ADPA System
-  **Planning** (4 documents)
 - Work Breakdown Structure
 - Project Management Plan
 - Risk Management Plan
 - Communication Plan
-  **Requirements** (3 documents)
 - Business Requirements Specification
 - Functional Requirements
 - System Requirements

Step 2: Single Document Conversion

Before: Raw Markdown

```
---
title: "Project Charter"
category: "project-charter"
author: "ADPA System"
version: "1.0"
---

# Project Charter: ADPA
```

Executive Summary

This Project Charter authorizes the initiation...

Project Objectives

Objective	Success Criteria	Timeline
-----	-----	-----
Automate Documentation	95% accuracy	Q2 2025

-

... [truncated]

=== ADOBE-CREDENTIALS-SETUP.MD (primary) ===

Path: ADPA\ADOBE-CREDENTIALS-SETUP.md

Relevance Score: 65

How to Get Your Adobe.io Credentials

Step 1: Access Adobe Developer Console

1. Go to <https://developer.adobe.com/console>
2. Sign in with your Adobe ID (the same one you use for Adobe Creative Cloud)

Step 2: Find or Create Your Project

If you already have a project:

1. Click on your existing project
2. Go to the **Credentials** section

If you need to create a new project:

1. Click **Create new project**

2. Give it a name like "ADPA Document Processing"
3. Click **Create**

Step 3: Add Adobe PDF Services API

1. In your project, click **Add API**
2. Find **Adobe PDF Services API** in the list
3. Click **Next**
4. Choose **Server-to-Server** authentication
5. Click **Save configured API**

Step 4: Get Your Credentials

After adding the API, you'll see your credentials:

Copy these values to your .env file:

```
# From the "Credentials" section in Adobe Developer Console:  
  
ADOBE_CLIENT_ID=your_client_id_from_console  
ADOBE_CLIENT_SECRET=your_client_secret_from_console  
ADOBE_ORGANIZATION_ID=your_org_id_from_console
```

Where to find each value:

- **ADOBE_CLIENT_ID**: Listed as "Client ID" in the credentials section
- **ADOBE_CLIENT_SECRET**: Listed as "Client Secret" (click "Retrieve client secret")
- **ADOBE_ORGANIZATION_ID**: Listed as "Organization ID" at the top of the console

Step 5: Test Your Credentials

Run this test to make sure your credentials work:


```
curl -X POST "https://ims-na1.adobelogin.com/ims/token" \  
-H "Content-Type: application/x-www-form-urlencoded" \  
-d "grant_type=client_credentials&client_id=YOUR_CLIENT_ID&client_se
```

If successful, you'll get back an access token!

Step 6: Update Your .env File

1. Open your `.env` file in the ADPA directory
2. Replace the placeholder values with your actual credentials:

```
ADOBE_CLIENT_ID=abcd1234efgh5678    # Your actual Client ID  
ADOBE_CLIEN  
... [truncated]  
  
=== CLI-REFACTOR-IMPLEMENTATION-GUIDE.MD (documentation) ===  
Path: CLI-REFACTOR-IMPLEMENTATION-GUIDE.md  
Relevance Score: 62  
  
# CLI Refactor Implementation Guide  
  
This guide outlines the recommended steps to refactor the Requirements  
  
---  
  
## 1. Adopt a CLI Framework (Yargs)  
  
**Why:**  
- Simplifies argument parsing and validation  
- Automatically generates help output  
- Makes commands and options declarative and discoverable  
  
**How:**  
- Install Yargs:  
  ``sh  
  npm install yargs @types/yargs
```

- Refactor `src/cli.ts` to use Yargs for all command and option parsing.
- Example starter:

```
import yargs from 'yargs';
import { hideBin } from 'yargs/helpers';

yargs(hideBin(process.argv))
  .command('generate [key]', 'Generate a document', (yargs) => {
    yargs.positional('key', { type: 'string', describe: 'Document'
  }, (argv) => {
    // Call generate logic
  })
  .option('output', { type: 'string', default: 'generated-document'
  .help()
  .argv;
```

- Remove all manual `process.argv` parsing and helper functions.

Common Issues & Solutions:

- **Duplicate imports:** Remove old import statements when adding Yargs imports
- **Type errors:** Use proper type assertions for argv values (e.g., `argv.format as 'markdown' | 'json' | 'yaml'`)
- **Missing modules:** Create placeholder functions or use dynamic imports for non-essential commands during transition

2. Increase Modularity (Command Extraction)

Why:

- Easier to maintain and extend
- Each command is independently testable

How:

- Create a `src/commands/` directory.

- Move logic for each major command (e.g., generate, confluence, sharepoint, vcs) into its own file:

```
src/  
  commands/  
    generate.ts  
    confluence.ts  
    sharepoint.ts  
    vcs.ts  
    utils/  
      validation.ts
```

... [truncated]

=== STEP-2-COMPLETION-SUMMARY.MD (other) ===

Path: STEP-2-COMPLETION-SUMMARY.md

Relevance Score: 51

Step 2 Implementation Complete: Increased Modularity (Command Extraction)

What Was Accomplished

Command Modules Created

- `src/commands/confluence.ts` - Confluence integration commands
- `src/commands/sharepoint.ts` - SharePoint integration commands
- `src/commands/vcs.ts` - Version Control System commands
- `src/commands/utils/validation.ts` - Input validation utilities
- `src/commands/utils/common.ts` - Shared command utilities

Command Structure Implemented

src/		
commands/		
analyze.ts	✓	(existing)
confluence.ts	✓	(new)
generate.ts	✓	(existing, enhanced)
index.ts	✓	(updated)
setup.ts	✓	(existing)
sharepoint.ts	✓	(new)
status.ts	✓	(existing)
validate.ts	✓	(existing)
vcs.ts	✓	(new)
utils/		
common.ts	✓	(new)
validation.ts	✓	(new)

New CLI Commands Added

Confluence Commands

- `rga confluence init` - Initialize Confluence configuration
- `rga confluence test` - Test Confluence connection
- `rga confluence publish` - Publish documents to Confluence
- `rga confluence status` - Show Confluence integration status
- `rga confluence oauth2 login` - Start OAuth2 authentication
- `rga confluence oauth2 status` - Check OAuth2 authentication status
- `rga confluence oauth2 debug` - Debug OAuth2 authentication

SharePoint Commands

- `rga sharepoint init` - Initialize SharePoint configuration
- `rga sharepoint test` - Test SharePoint connection
- `rga sharepoint publish` - Publish documents to SharePoint
- `rga sharepoint status` - Show SharePoint integration status
- `rga sharepoint oauth2 login` - Start OAuth2 authentication
- `rga sharepoint oauth2 status` - Check OAuth2 authentication status

- `rga sharepoint oauth2 debug` - Debug OAuth2 authentication

VCS Commands

- `rga vcs init` - Initialize Git repository
... [truncated]

project starts with clear objectives, comprehensive planning, secure and scalable architecture considerations, stakeholder alignment, and risk preparedness. It aligns tightly with PMBOK standards and the project's detailed user stories and technical context.