

# Using 4D AIKit in Modern 4D Applications

By Soukaina BACHIKH, Customer Success Engineer, 4D Inc.

Technical Note 26-01

## Table of Contents

Abstract .....	3
Introduction .....	3
Overview of 4D AIKit .....	3
Component overview and capabilities .....	3
Compatibility and setup .....	4
Configuration parameters .....	5
Implementing AI Features .....	6
Architecture and Best Practices .....	6
Configuration Management .....	6
Managing AI Calls and Contexts .....	6
Demo: Intelligent Business Assistant .....	8
Demo Overview .....	8
Database Schema .....	9
AI Configuration .....	9
Document Data Extraction .....	11
Generating Summaries and Insights .....	13
AI Chat Assistant .....	15
Integration with 4D Forms .....	18
Getting Started .....	19
Deployment Considerations: .....	20
Token Usage and Costs .....	20
Security and Data Privacy .....	21
Conclusion .....	22
Additional Resources .....	22

## Abstract

As modern business solutions increasingly rely on intelligent automation and natural language processing, integrating AI into 4D applications has become both relevant and necessary. Introduced in 4D 20 R9, 4D AIKit is a built-in component that allows developers to easily connect 4D to powerful large language and vision models through a native interface.

This technical note introduces the fundamentals of 4D AIKit, explains how to configure and use it, and demonstrates its practical value through an integrated business-oriented demo. The example shows how text, image, and conversational features can be combined to create practical AI-powered solutions. It also provides essential guidance on cost awareness, performance optimization, and deployment best practices.

## Introduction

Artificial Intelligence has transformed from a futuristic concept into a practical business tool. Organizations across industries are leveraging AI capabilities to automate processes, extract insights from data, and enhance user experiences. For 4D developers, the challenge has been integrating these powerful AI services into applications without extensive external dependencies or complex infrastructure.

4D AIKit addresses this need by providing a native, straightforward interface for connecting 4D applications to leading AI models. Whether building document processing systems, conversational interfaces, or intelligent data analysis tools, 4D AIKit simplifies the integration process while maintaining the robustness and reliability expected in enterprise applications.

## Overview of 4D AIKit

### Component overview and capabilities

4D AIKit is a built-in comprehensive component that provides three primary functional areas through the `cs.AIKit` namespace:

### Text Generation and Processing

The chat completions API enables human-like text generation for content creation, summarization, translation, and data transformation. This functionality supports generating reports, emails, product descriptions, or any text-based output required in business applications. The component maintains conversation context through message collections, enabling multi-turn dialogues with maintained state.

### Vision and Image Analysis

Vision capabilities extract information from images, documents, and visual content. This enables optical character recognition, object detection, and visual question answering

without requiring specialized image processing infrastructure. The vision helper class simplifies image analysis by combining chat completions with image input.

## Image Generation

The images API generates visual content from text descriptions, enabling applications to create illustrations, placeholders, or design concepts programmatically. This capability supports creative workflows and automated asset generation.

The 4D AIKit component is structured around several core classes that provide organized access to above AI capabilities:

Class	Purpose	Example Code
<b>OpenAI</b>	Main client class for API authentication and configuration. Entry point for all AIKit functionality.	<pre>\$aiClient := cs.AIKit.OpenAI.new("api-key")  // With config \$config := New object("model"; "gpt-4o-mini") \$aiClient := cs.AIKit.OpenAI.new("api-key"; \$config)</pre>
<b>OpenAIChatCompletionsAPI</b>	Handles text generation and chat-based AI interactions. Processes conversation messages and returns AI responses.	<pre>\$messages := New collection \$messages.push(New object("role"; "user"; "content"; "Explain AI")) \$result := \$aiClient.chat.completions.create(\$messages; \$options)</pre>
<b>OpenAIVision/ OpenAIVisionHelper</b>	Specialized helper for vision tasks, image analysis, OCR, visual question answering, and document extraction.	<pre>\$vision := \$aiClient.chat.vision.fromFile(\$file) \$params := New object("model"; "gpt-4o") \$result := \$vision.prompt("Extract text"; \$params)</pre>
<b>OpenAIImagesAPI</b>	Handles AI-powered image generation from text descriptions for creating visual assets.	<pre>\$prompt := "Business dashboard" \$options := New object("model"; "dall-e-3"; "size"; "1024x1024") \$result := \$aiClient.images.generate(\$prompt; \$options)</pre>
<b>OpenAIError</b>	Structured error object returned when API requests fail. Provides detailed error information.	<pre>\$result := \$aiClient.chat.completions.create(\$msgs; \$options)  If (Not(\$result.success))     \$error := \$result.errors     ALERT("Error: " + \$error[0].message) End if</pre>

## Compatibility and setup

## System Requirements

- ✓ 4D 20 R9 (already includes 4D AIKit as a built-in component) or later
- ✓ Internet connectivity for API calls to external AI providers
- ✓ Active API key from at least one supported AI provider
- ✓ A list of compatible providers is available via this link:  
<https://developer.4d.com/docs/aikit/compatible-openai>

**Note:** For 4D 21 please make sure to add the 4D AIKit component since it's no longer built-in

## Supported models

4D AIKit supports OpenAI models and models from any OpenAI-compatible API provider. The component is designed to work seamlessly with various AI providers through a unified interface.

Here's the link to OpenAI different models: <https://platform.openai.com/docs/models>

## Basic API Configuration

The AIKit component is initialized through the `cs.AIKit.OpenAI` class, requiring an API key and optional configuration parameters. The configuration supports custom endpoints for OpenAI-compatible services, parameter settings, and default model preferences.

```
// Basic configuration
var $aiClient : cs.AIKit.OpenAI
$aiClient:=cs.AIKit.OpenAI.new("your-api-key-here")

// Configuration with custom settings
var $config : Object
$config:=New object
$config.model:="gpt-4o-mini"
$config.temperature:=0.7
$config.maxTokens:=1000

$aiClient:=cs.AIKit.OpenAI.new("your-api-key-here"; $config)
```

## Configuration parameters

- **model:** used to choose the AI model based on the purpose (text generation, image generation, and etc.)
- **temperature:** Used to control the randomness and creativity of responses - lower values produce consistent outputs, higher values generate more varied responses
- **maxTokens:** Used to limit the maximum length of the generated response and control API costs
- **timeout:** Used to set the maximum wait time for API responses before terminating the request

And much more specified in the documentation

**Note:** Before getting an API Key from cloud AI provider, a local test might be done using Local Provider like Ollama, please check this tech tip for more details : [Testing 4D AIKit Without an OpenAI API Key \(https://kb.4d.com/assetid=79903\)](https://kb.4d.com/assetid=79903)

## Implementing AI Features

### Architecture and Best Practices

Successful AI integration in 4D applications follows established architectural patterns that promote maintainability, reliability, and scalability.

#### Class-Based Architecture

Organizing AI functionality into dedicated classes provides clear separation of concerns. Each class handles a specific domain: document analysis, summarization, conversation management, or configuration. This approach encapsulates AI logic, making code easier to test, maintain, and extend.

#### Asynchronous Processing Pattern

AI API calls can take several seconds to complete. Processing these calls in worker processes prevents UI freezing and maintains application responsiveness. The pattern involves launching a worker process for AI operations, updating the database with results, and using timer-based polling in the UI process to reflect changes.

**Note:** 4D AIKit provides a native asynchronous call pattern using callbacks (see [Asynchronous Call \(https://developer.4d.com/docs/aikit/asynchronous-call\)](https://developer.4d.com/docs/aikit/asynchronous-call)). However, this approach only works within the current process. For multi-process cases, use CALL WORKER or CALL FORM instead.

#### Configuration Management

Centralizing AI configuration in a dedicated class allows consistent access to API keys, model selections, and parameters across the application. Configuration can be loaded from Storage, settings file, or secure vaults, providing flexibility for different deployment scenarios.

### Managing AI Calls and Contexts

Effective AI integration requires careful management of conversation contexts, message histories, and system prompts.

#### Message History Management

AI models process conversations as collections of messages with specific roles: system, user, and assistant. The system message establishes behavior and context, user messages contain queries or instructions, and assistant messages represent AI responses. Maintaining message history enables contextual conversations where the AI remembers previous exchanges.

```
// Building conversation context
var $messages : Collection
$messages:=New collection

// System message provides context
$messages.push(New object(\
  "role"; "system"; \
  "content"; "You are an assistant analyzing business documents."))

// Add conversation history
For each ($msg; $messageHistory)
  $messages.push(New object(\
    "role"; $msg.role; \
    "content"; $msg.content))
End for each

// Add current user message
$messages.push(New object(\
  "role"; "user"; \
  "content"; $userMessage))

// Call AI with full context
$result=$aiClient.chat.completions.create(New object(\
  "model"; "gpt-4o-mini"; \
  "messages"; $messages))
```

**Note:** The difference between the message roles is explained in this Tech Tip: [Understanding “roles” in AI Conversations \(https://kb.4d.com/assetid=79838\)](https://kb.4d.com/assetid=79838)

## Prompt Engineering

The quality of AI responses depends significantly on prompt design. Effective prompts are specific, provide clear instructions, include relevant context, and specify desired output formats. Structured prompts with explicit formatting requirements yield more predictable and usable results.

## Error Handling

Robust error handling ensures applications remain stable when AI services encounter issues.

Common Error Scenarios:

- API authentication failures due to invalid or expired keys

- Network connectivity issues affecting API requests
- Rate limiting when request volumes exceed API quotas
- Invalid request parameters causing API rejections
- Token limit exceeded errors when input or output exceeds model capacity

4D AIKit returns a structured `OpenAIError` object whenever an API request fails.

Example:

```
var $client:=cs.AiKit.OpenAI.new("API Key")
var $messages : Collection
var $options : Object
var $result : Object

$messages:=[{role: "user"; content: "How can I use 4D AIKit?"}]

$options:={model: "gpt-4o-mini"; max_tokens: 100}

$result:=$client.chat.completions.create($messages; $options)

If (Not($result.success))
    $error:=$result.errors
    ALERT("Error: "+$error[0].message)
End if
```

## Demo: Intelligent Business Assistant

### Demo Overview

The Demo-AiKit application demonstrates practical AI integration patterns through an intelligent document management system. The application combines three core AI capabilities: vision-based document analysis, intelligent summarization, and conversational interfaces.

### Application Features

- **AI-Powered Document Analysis:** Automatically extracts structured data from business documents including invoices, receipts, and contracts using GPT-4o vision capabilities.
- **Intelligent Summarization:** Generates multiple summary types (brief, detailed, executive, key points) tailored to different business needs.
- **Document Chat Interface:** Enables natural language conversations about document content with full context awareness.
- **Asynchronous Processing:** Handles long-running AI operations without blocking the user interface.

### Architecture Components



- DocumentAnalyzer: Handles document analysis and vision-based data extraction
- SummaryGenerator: Creates AI-powered summaries in multiple formats
- ConversationManager: Manages document-focused chat conversations
- AIConfig: Centralizes AI configuration and credentials

## Database Schema

- Document: Stores uploaded files with metadata and processing status
- ExtractedData: Holds structured data extracted from documents
- Summaries: Maintains generated summaries of different types
- Conversation: Tracks chat message history and conversation state

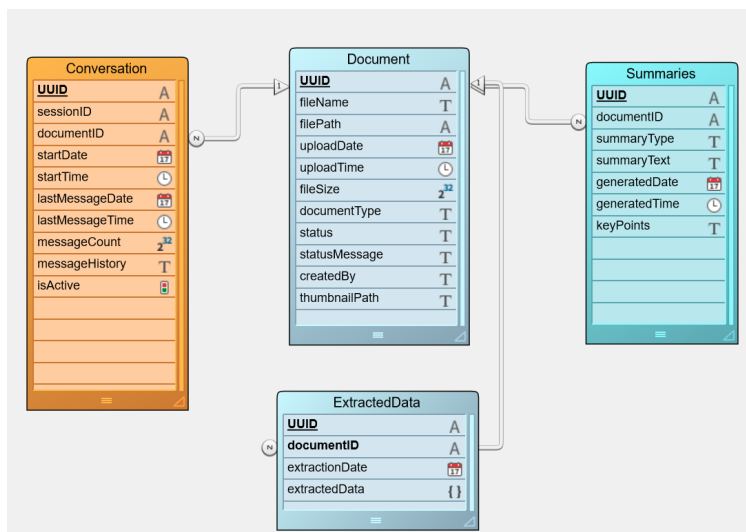


Figure 1 - Demo Database Structure

## AI Configuration

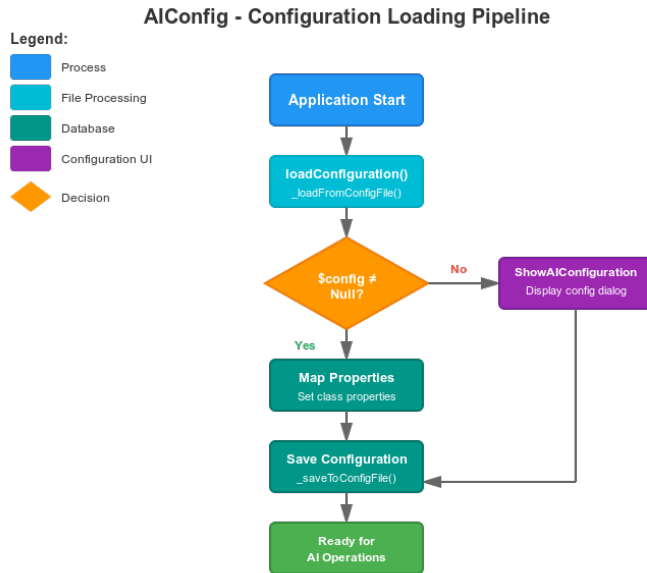
A centralized configuration management singleton class that stores and provides AI-related settings across the application. Acts as a single source of truth for API credentials, model preferences, and default parameters.

The AIConfig class must be instantiated during application startup to ensure configuration is loaded before any AI operations.

Add this to your On Startup database method:

```
// On Startup
var $aiConfig : cs.AIConfig
$aiConfig:=cs.AIConfig.me // Initialize singleton and load configuration
```

## Configuration Flow



**Figure 2 - Configuration Loading Workflow**

## Class Implementation

### Main Method

- shared singleton Class constructor() - Singleton initialization, auto-loads configuration
- loadConfiguration() - Loads settings from file or triggers setup dialog
- getClient() - Returns configured AI client instance

### Helper Methods

- \_saveToConfigFile() - Persists configuration to aiconfig.json
- \_loadFromConfigFile()- Reads and parses configuration file

## Configuration Dialog

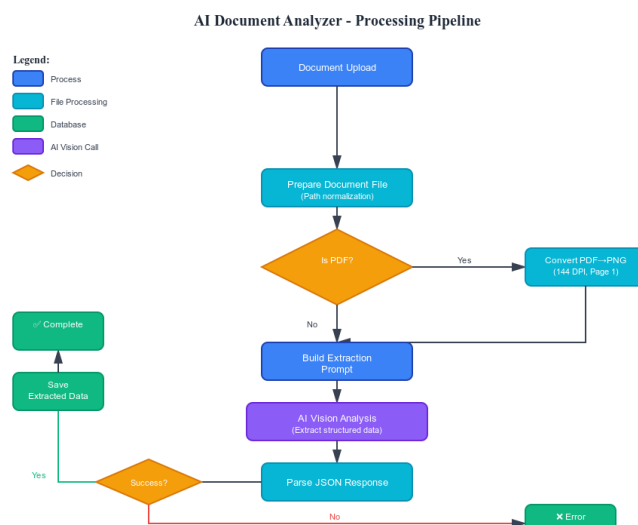
The image shows a software configuration window titled "AI Configuration". It contains several input fields and two buttons at the bottom. The fields are: "Provider\*" with the value "OpenAI", "API Key\*" with the value "sk-proj-", "Base URL" with the value "eg. http://127.0.0.1:11434/v1", "Default Model" with the value "gpt-4o-mini", "Vision Model" with the value "gpt-4o", "Max Tokens" with the value "2500", and "Temperature" with the value "0.3". The "Save" and "Cancel" buttons are located at the bottom right of the window.

**Figure 3 - Configuration Form**

## Document Data Extraction

The document analysis module uses GPT-4o vision capabilities to extract structured information from various document types (Receipt, Invoice, etc.) without requiring templates or predefined formats.

## Extraction Process



**Figure 4 - Analyzing Document Workflow**

## Class Implementation

### Main Method

- analyzeDocument(docID : Text) - Main entry point, orchestrates the entire pipeline

### Helper Methods

- \_extractGenericData() - Coordinates file prep, AI call, and data saving
- \_prepareDocumentFile() - Path normalization & PDF→PNG conversion
- \_convertPdfToImage() - PDF to PNG conversion using pdfium plugin at 144 DPI
- \_buildExtractionPrompt() - Constructs the AI prompt
- \_analyzeDocumentWithAI() - Makes the vision API call
- \_saveExtractedData() - Persists results to database

#### Note:

1. The \_convertPdfToImage requires the 4d-plugin-pdfium developed by “Keisuke Miyako” to be added into the Plugin folder. To download the plugin, please click this link : <https://github.com/miyako/4d-plugin-pdfium/releases/tag/universal-binary>
2. From 4D 21 R2 onwards, native support for uploading and using files directly in AI requests is available, here is how to use it : [Using PDF Files with 4D AIKit \(Purpose="assistants"\)](https://kb.4d.com/assetid=79910) (<https://kb.4d.com/assetid=79910>)

### Prompt Engineering

Here's the used prompt to analyze the document:

You are analyzing a document. Extract ALL relevant information you can find:

1. Document type (e.g., Invoice, Receipt, Contract, Letter, Report, etc.)
2. Document title or subject
3. Document date (format: MM-DD-YYYY)
4. Main content summary (2-3 sentences)
5. Key entities (names, organizations, amounts, dates, etc.)
6. Any other relevant fields specific to this document type

Return ONLY valid JSON. Include ALL fields you can extract.

Required keys: documentType, title, documentDate, summary, keyEntities

Add any other relevant fields based on document type.

DO NOT include ```json or ``` markers.

DO NOT include any text before or after the JSON.

### Prompt Design Principles

- Clear structure - Numbered list of extraction requirements
- Format specification – (MM-DD-YYYY) for dates ensures consistency
- Flexibility - "Any other relevant fields" allows type-specific extraction
- Zero-shot learning - Works across document types without examples
- Output constraints - Explicitly prevents markdown and extra text

## Why the strict constraints?

AI models often add helpful context like "Here's the extracted data:" or wrap JSON in json blocks. These break JSON Parse(), so prevent them upfront rather than post-processing.

## The AI Call

```
Function _analyzeDocumentWithAI($file : 4D.File; $prompt : Text) -> $result : Object
```

```
// Initialize vision helper with document file
$visionHelper:=This.client.chat.vision.fromFile($file)

// Configure API parameters
$params:={\
  model: This.config.visionModel; \ // The used model is "gpt-4o"
  max_tokens: 1000; \ // Enough for structured data
  temperature: 0.1} // Low = deterministic extraction

// Execute vision analysis
$result:=$visionHelper.prompt($prompt; $params)
return $result
```

## Configuration

- Temperature 0.1 - Ensures consistent, deterministic extractions (not creative responses)
- Max tokens 1000 - Balances API cost with sufficient response length for complex documents
- Vision model - Handles both images and text extraction from document scans/photos

## Generating Summaries and Insights

The summarization engine creates tailored summaries based on extracted document data, providing different perspectives suited to various business needs.

## Class Implementation

### Main Method

- generateSummary(docID, summaryType) - Main entry point, generates AI-powered document summaries

### Helper Methods

- \_validateDocumentData() - Validates document and extracted data existence
- \_buildSummaryPrompt() - Constructs type-specific prompts with HTML formatting instructions
- \_generateWithAI() - Makes the AI completion call

- `_saveSummary()` - Persists generated summary to database

## Summary Types & Prompt Engineering

The class supports 4 distinct summary types, each with tailored prompts:

### 1. Brief Summary (2-3 sentences)

- Focus: Most important information only
- Format: Simple HTML with inline CSS

### 2. Detailed Summary (Comprehensive)

- Focus: All key details with structured sections
- Sections: Document Overview, Key Parties, Important Dates & Amounts, Notable Items
- Format: Semantic HTML with `<h3>`, `<ul>`, `<strong>` tags

### 3. Executive Summary (Management-focused)

- Focus: Bottom line, key financials, action items
- Visual: Colored priority badges, highlighted boxes
- Format: Blue info boxes for critical items, priority color coding

### 4. Key Points (Actionable items)

- Focus: Action-required items with priority levels
- Format: Card-based layout with:
- Priority badges (High=#ef4444, Medium=#f59e0b, Low=#10b981)
- Category tags (Financial/Deadline/Compliance/General)
- Recommended actions

## Common Prompt Pattern

[Task description]

Return as clean HTML with inline CSS styling. [Specific structure]  
DO NOT include ``html or `` markers.

Document information: [Context from extracted data]

## The AI Call

**Function** `_generateWithAI($prompt : Text) -> $result : Object`

**var** `$messages : Collection`

**var** `$params : Object`

`$messages := New collection`

`$messages.push(New object("role"; "system"; "content"; "You are a business document analyst."))`

```

$messages.push(New object("role"; "user"; "content"; $prompt))

$params:=New object(\
  "model"; This.config.defaultModel; \
  "max_tokens"; This.SUMMARY_MAX_TOKENS; \
  "temperature"; This.SUMMARY_TEMPERATURE)

$result:=This.client.chat.completions.create($messages; $params)

return $result

```

## Configuration

- System role: "You are a business document analyst" - Sets professional context
- SUMMARY\_MAX\_TOKENS: 800 - Sufficient for detailed summaries
- SUMMARY\_TEMPERATURE: 0.3 - Balanced between consistency and natural language

## HTML Rendering

Summaries are generated with clean HTML formatting including semantic tags and inline CSS styling. The HTML output renders beautifully in web areas, providing a professional presentation without external stylesheets.

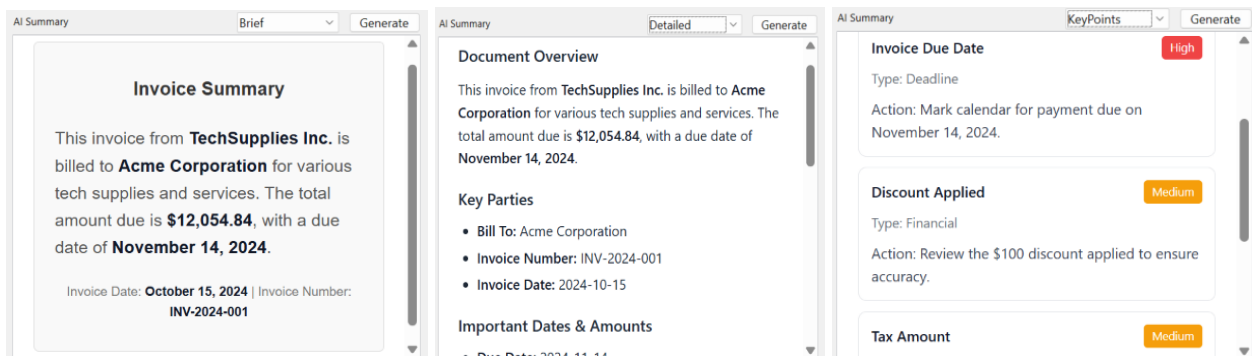
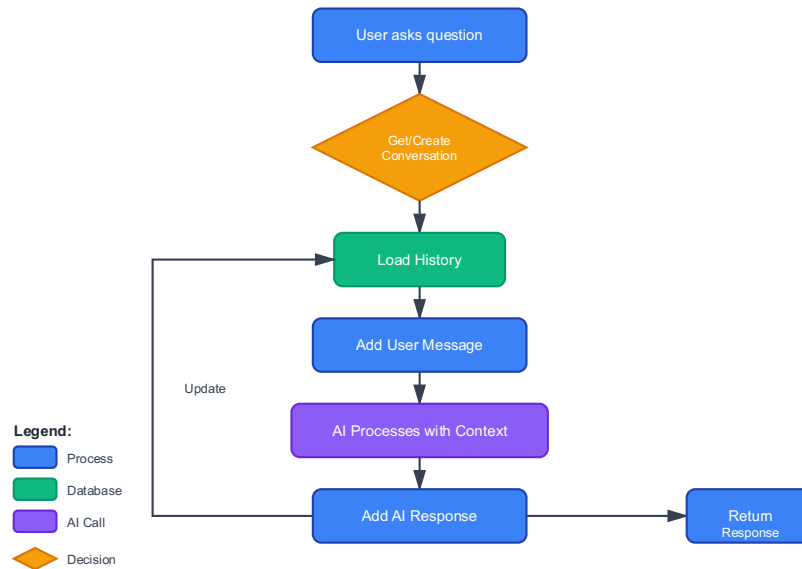


Figure 5 - Examples of Different Summary Types

## AI Chat Assistant

The Chat Assistant is the component where users can interact with the AI to ask questions about the selected document.

## Conversation Flow



**Figure 6 - Conversation Workflow**

## Class Implementation

### Main Method

- `sendMessage(docID, userMessage)` - Main entry point, handles conversational Q&A about the selected document

### Helper Methods

- `_getOrCreateConversation()` - Retrieves existing or creates new conversation
- `_createNewConversation()` - Initializes conversation with system context
- `_buildSystemMessage()` - Constructs document-aware system prompt
- `_updateConversation()` - Persists message history and metadata
- `_callAI()` - Makes the AI completion call with conversation history

## System Message Construction

The system message provides AI with document context:

You are a helpful assistant that answers questions about business documents.  
 You have access to the following document information:  
 Document: [filename] (Type: [type])

Extracted Data:  
 [Complete document context from BuildDocumentContext()]

Answer questions based on this information. If information is not available,  
 say so rather than making assumptions. Be concise and professional.



Why do these instructions matter?

- AI knows it's working with a specific document
- Has access to all extracted data upfront
- Explicitly instructed to avoid hallucination<sup>1</sup> ("say so rather than making assumptions")
- Professional tone enforced

Example: If the Chat Assistant receives a question unrelated to the document, the AI declines to answer even if the model knows the information. This is because the system prompt specifically instructs it to only answer questions related to the document, as shown below:

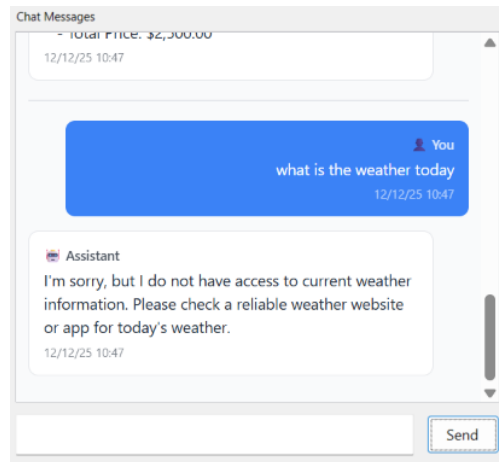


Figure 7 - Example of Chat Assistant responses to document unrelated questions

## The AI Call

```
Function _callAI($history : Collection)->$result : Object
var $params : Object

$params={\
  model: This.config.defaultModel; \
  max_tokens: This.MAX_TOKENS; \
  temperature: This.TEMPERATURE}

$result:=This.client.chat.completions.create($history; $params)

return $result
```

## Configuration

<sup>1</sup> **Hallucination** is when a generative AI model produces confident-sounding but factually incorrect, nonsensical, or fabricated information instead of retrieving verified information

- MAX\_TOKENS: 1000 - Allows detailed answers while controlling cost
- TEMPERATURE: 0.5 - Balanced between factual accuracy and conversational fluidity

## Integration with 4D Forms

The DocumentManager form provides a complete user interface integrating the most used AI capabilities through intuitive controls and real-time feedback.

### UI Components:

- Document List: Selection list with upload, analyze and delete capabilities
- Document Preview Area: Web area displaying document content
- Extracted Data Panel: Formatted text showing AI-extracted information with an edit form, in case the information was not well retrieved by the AI
- Summary Display: HTML-rendered summaries in web area
- Chat Interface: Interactive conversation UI with message history

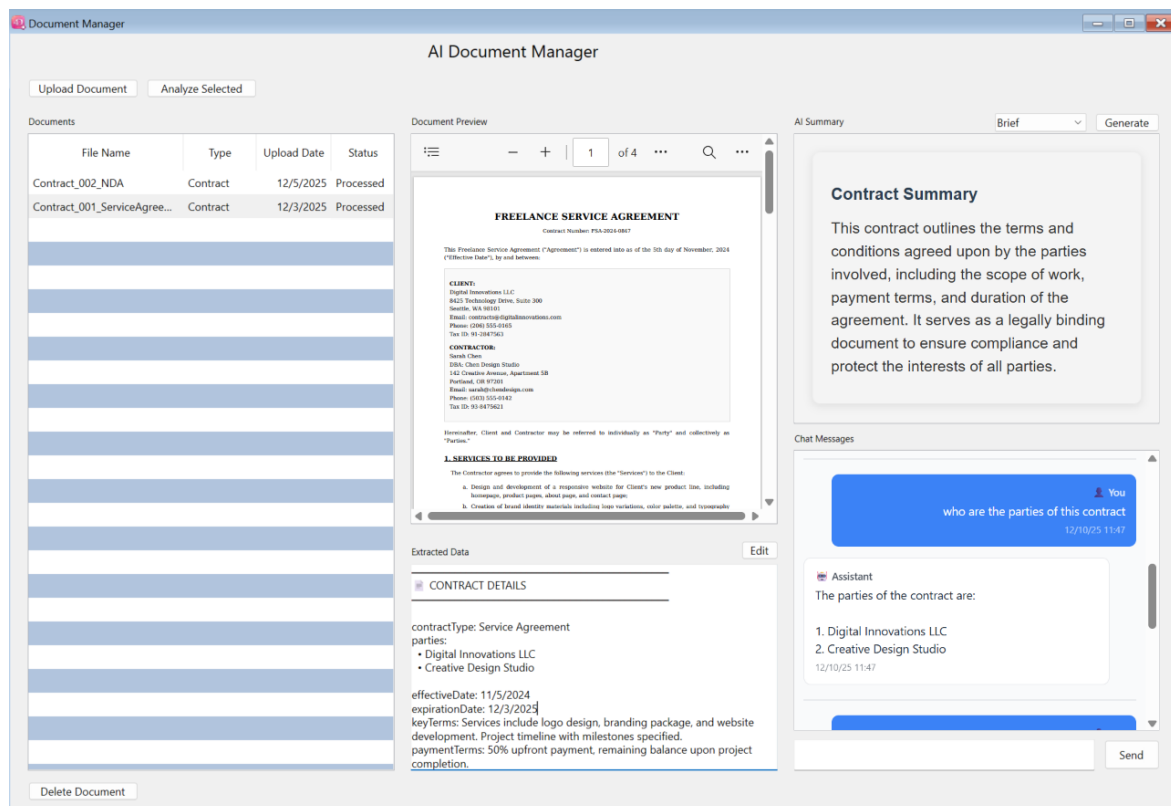


Figure 8 - Document Manager Interface

Edit the extracted data below. The data must be valid JSON format.

```

{
  "documentType": "Invoice",
  "title": "Invoice",
  "documentDate": "2024-10-15",
  "summary": "This invoice from TechSupplies Inc. is billed to Acme Corporation for various tech supplies and services. The total amount due is $12,054.84, with a due date of November 14, 2024.",
  "keyEntities": {
    "billTo": "Acme Corporation",
    "invoiceNumber": "INV-2024-001",
    "invoiceDate": "2024-10-15",
    "dueDate": "2024-11-14",
    "totalAmount": "$12,054.84",
    "items": [
      {
        "description": "Dell Latitude 5520 Laptop",
        "quantity": 5,
        "unitPrice": "$1,299.00",
        "totalPrice": "$6,495.00"
      },
      {
        "description": "HP LaserJet Pro M404dn Printer",
        "quantity": 2,
        "unitPrice": "$349.99",
        "totalPrice": "$699.98"
      },
      {
        "description": "3-Year Extended Warranty",
        "quantity": 5,
        "unitPrice": "$199.99"
      }
    ]
  }
}

```

Save Cancel

Figure 9 - Extracted Data Edit Form

## User Workflow

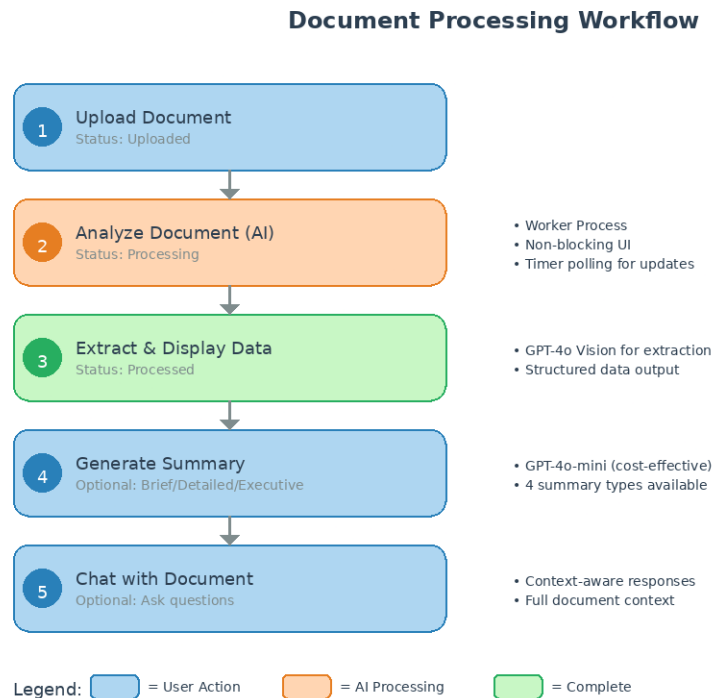


Figure 10 - Document Processing Workflow - Complete user journey from upload to analysis

## Getting Started

1. Download the demo project attached to this Technical Note
2. System Requirements: Open this project under 4D 20 R9 or later
3. Install AIKit Component (4D 21 only):
  - Place the component in the Components folder of your project
  - Note: 4D 20 R9 includes AIKit natively and does not require this step
4. Configure AI Settings:
  - On first launch, if no configuration exists, the AI Configuration dialog will appear automatically
  - Enter your API key and configure your preferred AI provider settings
  - If configuration is already present, the application will launch directly the Document Manager
  - You can also manually edit the “aiconfig.json” file in the database folder with valid credentials
5. Launch the application: Run the **Document Manager** form to test the application (or it will launch automatically if already configured)
6. Upload a document: Upload a business document or any other document (note: prompts are optimized for business documents like invoices, receipts, contracts, and etc.)
7. Analyze the document: Click on the uploaded file and click the “**Analyze Document**” button
8. Generate summaries: Use the Summary section to generate document summaries
9. Chat with AI: Start chatting with the AI Chatbot about the selected document

## Deployment Considerations:

### Token Usage and Costs

AI services operate on a pay-per-use model based on token consumption. Understanding token economics is essential for cost-effective deployment.

### Token Basics

Tokens represent pieces of text, roughly equivalent to 4 characters or 0.75 words in English. Both input (prompts) and output (responses) consume tokens. Vision models charge based on image resolution in addition to text tokens.

### Cost Optimization Strategies

- Model Selection: Use gpt-4o-mini for routine tasks, reserving gpt-4o for complex analysis requiring vision or advanced reasoning
- Prompt Optimization: Craft concise prompts that provide necessary context without redundant information
- Token Limits: Set maxTokens parameters to prevent unexpectedly long responses

## Model Selection in Demo-AIKit

- Vision Analysis: GPT-4o (required for image understanding)
- Summaries: GPT-4o-mini (cost-effective for text generation)
- Chat: GPT-4o-mini (balanced performance and cost)

## Security and Data Privacy

AI integration introduces specific security considerations beyond standard application security.

### API Key Security

- Never embed API keys in source code or version control
- Store keys in secure configuration files, environment variables, or encrypted storage
- Use separate keys for development, testing, and production environments

### Data Privacy

Data sent to AI services leaves the local environment. Organizations must consider regulatory requirements like GDPR, HIPAA, or industry-specific compliance when processing sensitive information. Options include data anonymization before AI processing...

Check this link for more information: <https://trust.openai.com/>

### File Processing Limits

Demo-AIKit enforces a 10MB file size limit for vision processing, balancing processing speed with reasonable business document sizes. Applications can implement additional validation for file types, scanning for malware, or content filtering before AI processing.

## Troubleshooting

Common Issues and Solutions:

Issue Category	Symptom	Solution
Connection Failures	API calls fail with authentication errors	Verify API key validity; ensure key not expired/revoked; check network allows HTTPS requests to OpenAI endpoints.
Token Limit Exceeded	Error indicates context length exceeded	Reduce prompt size; delete conversation history; switch to models with larger token limits.

Slow Processing	AI operations take excessive time	Check network latency; verify no rate-limit queuing; use faster model variants.
PDF Conversion Issues	Document analysis fails for PDF files	Verify pdfium plugin installation; confirm PDF is valid/not protected; check disk space; reduce DPI if memory issues occur.
Unexpected AI Responses	AI generates incorrect or inconsistent results	Improve prompt clarity; lower temperature; add explicit output format; ensure extracted data is correctly passed.

## Conclusion

4D AIKit provides a powerful, native solution for integrating artificial intelligence into 4D applications. By offering a straightforward interface to leading AI models, the component enables developers to add sophisticated capabilities without managing complex external dependencies or infrastructure.

The Demo-AIKit application demonstrates practical integration patterns that combine vision-based document analysis, intelligent summarization, and conversational interfaces into a cohesive business solution. The architectural approaches illustrated class-based organization, asynchronous processing, and centralized configuration provide templates applicable to diverse AI integration scenarios.

Success with AI integration extends beyond technical implementation. Cost awareness through thoughtful model selection, performance optimization, and robust error handling ensure production-ready solutions. Security considerations, particularly around API key management and data privacy, require careful attention in enterprise deployments.

## Additional Resources

Compatible Provider: <https://developer.4d.com/docs/aikit/compatible-openai>

OpenAI Models: <https://platform.openai.com/docs/models>

Testing 4D AIKit Without an OpenAI API Key: <https://kb.4d.com/assetid=79903>

Asynchronous Call: <https://developer.4d.com/docs/aikit/asynchronous-call>

Understanding “roles” in AI Conversations: <https://kb.4d.com/assetid=79838>

Pdfium Plugin: <https://github.com/miyako/4d-plugin-pdfium/releases/tag/universal-binary>

Using PDF Files with 4D AIKit (Purpose="assistants"): <https://kb.4d.com/assetid=79910>