

AI Rules Checker

AI Rules Checker is a modular framework designed to bridge the gap between structured business processes and unstructured document intelligence. By allowing end users to create multiple sets of complex business rules written in natural language, we can transform traditional back-office tasks into "Self-Governing" workflows. This solution moves beyond simple OCR and key value pair extraction; it employs an architecture that interprets complex business requirements, justifies its decisions through auditable analysis, and intelligently selects the best extraction tools (Vision vs. Text) based on the specific needs of each check. Whether deployed as a standalone case management tool or embedded as a utility service within an existing application, this framework provides a standardized, scalable pattern for implementing high-trust AI across the enterprise.

1. Rules Engine Architecture

The **Rules Engine Type** determines where the "Source of Truth" for the workflow resides.

Option	Developer Guidance
Standalone	The "Full-Service" Model. Use this when the solution is the primary application. It manages the entire lifecycle, from ingestion to human-in-the-loop (HITL) routing and final closure using built-in UIs.
Embedded	The "Utility" Model. Use this when integrating into an existing Appian application. The engine acts as a stateless "Black Box" that receives data and returns structured check results. Your wrapper application is responsible for the process model, task routing, and data persistence.

2. Confidence Scoring

This toggle defines the granularity of the AI's self-assessment.

- **Disabled:** The AI only returns the `agentPass` boolean.
- **Scale:** Returns a qualitative value (e.g., "High", "Medium", "Low"). Best for human reviewers who need a quick "at-a-glance" feel for the AI's certainty.
- **Numerical:** Returns a specific score (e.g., 0-100). Use this if you plan to build **automated gateways** in Appian (e.g., "If confidence < 85, automatically route to Senior Reviewer").

3. AI Type Override (Advanced Settings)

This is the most critical technical toggle. It changes the **Agent Orchestration Pattern** and dictates how the AI interacts with Appian's "ARC" (Appian Runtime Component) tools.

A. SINGLE_PROMPT

- **The "All-in-One" Approach:** Passes all document text directly into the prompt context in one go.
- **Best Use Case:** High-speed processing of small files (<50 pages total).
- **Developer Note:** There are no "Tool Calls" here. Only use this if you are certain the document volume is low and text-based.

B. AGENT

- **The "Research Architect":** This mode uses a multi-step "Reconnaissance & Planning" phase. It doesn't read everything at once; it looks at the metadata first, then decides which tool (Vision vs. Text) is appropriate for each page.
- **Best Use Case:** "Messy" data. Use this for large PDFs (>50 pages), handwritten forms, or documents where checkboxes/signatures (Vision) are required.
- **Developer Note:** This is the most "expensive" in terms of time and tokens, but the most robust. It utilizes **Chain of Thought** reasoning to handle complex business logic.

C. AGENT_EXTRACTION_JUDGE

- **The "Verification Layer":** This mode treats upstream data (ex: DocCenter extractions) as the primary source. It only "breaks the glass" and calls additional tools if it finds a contradiction or missing mandatory data.
- **Best Use Case:** High-efficiency workflows where OCR has already happened. It prevents "re-reading" costs while maintaining a safety net for accuracy.
- **Developer Note:** Use this to reduce latency. It shifts the AI's role from a *worker* (finding data) to a *judge* (verifying data against rules).

4. Execution Hint

The **Execution Hint** is a high-level strategic instruction passed to the AI. Think of this as a "Manager's Note" that overrides the AI's default behavior.

- **Usage:** Use this to force a specific behavior without changing the underlying code.
- **Example:** If using **AGENT** mode, but you know the files are high-quality, you might add an Execution Hint: *"Prioritize ARC Read Bulk Documents for all files to speed up analysis; only use Vision if a signature is missing."*

Implementation Checklist for Developers

1. **Check Volume:** If >100 pages total, avoid **SINGLE_PROMPT**.
2. **Check Visual Requirements:** If the business check says "Check if signed," you **must** use one of the **AGENT** modes

5. Batch Type Override

This setting controls how the list of Checklist Items (Checks) is partitioned before being sent to the AI.

Option	Developer Guidance	Best For
SINGLE BATCH	Sends every single check in the Case Type to the AI in one massive payload.	Small Case Types or where the AI needs a holistic view of all requirements at once.
FIXED BATCH SIZE	Splits the checklist into equal chunks (e.g., 5 checks at a time).	Standardized processing. Use this to prevent "Middle-of-the-Prompt" loss where the AI ignores checks placed in the center of a long list.
INTELLIGENT BATCH	The system groups checks based on document dependency or logical similarity.	Complex Case Types. This ensures that all "Financial" checks or all "Identity" checks are processed together, reducing redundant document scanning.

6. Authoring "Prompt Instructions"

When creating the **Checklist Items** within a Case Type, the **promptInstructions** field must be written as **Deterministic Directives**.

To get the best performance from the LLM, developers should treat this field as "Business Logic in Natural Language."

The "Golden Rule" of Instructions

Avoid "vague" requests like "*Look for the date.*" Instead, use the **Scenario-Based** framework:

- **Positive Scenario (PASS IF):** Define the exact conditions for success.

- *Example:* "PASS IF the 'Effective Date' on the COI is within the last 12 months and matches the date on the Master Agreement."
- **Negative Scenario (FAIL IF):** Define common "gotchas."
 - *Example:* "FAIL IF the signature is present but the date field next to it is blank."
- **Default Assumption:** Tell the AI what to do when information is missing.
 - *Example:* "If no 'Expiration Date' is found, assume the document is valid for 1 year from the 'Issue Date'."

Formatting Template for Developers

Encourage your team to follow this structure for every check:

VERIFY THAT: [Primary Goal]

PASS IF

- [Specific Condition A]
- [Specific Condition B]

FAIL IF

-
-

RETURN NULL IF (if applicable)

-
-

DEFAULT: If [Data] is missing, [Default Action].

7. Procedure Documents

These are global documents added to the caseType that supplement the checks to perform with additional information. For example, you may have a check that's written as

PASS IF

- Criteria 1
- Criteria 2
- All additional criteria in <Procedure Document> are satisfied

.The system will prioritize full reads of procedure documents. They can be set to USE_WHEN = ALWAYS or USE_WHEN = CERTAIN_CENARIOUS (describe in plain text)