Icertis Integration Development Guidelines

Status	IN PROGRESS
Owner	RAMTEK-ext, Vaibhav
Stakeholders	

Best Practices and Guidelines for Custom Integration Development in Icertis

Introduction

Custom integrations in Icertis Contract Intelligence (ICI) extend the platform's capabilities to connect with external systems—such as ERP, CRM, e-signature, or analytics tools.

These integrations often handle critical business events like agreement publish, execution, or status change, so a structured approach is vital to ensure consistency, error recovery, and maintainability across implementations.

This document captures the recommended patterns, coding standards, and operational guidelines to follow when developing or maintaining custom integration tasks within Icertis.

Scope and Objectives

Scope

This document applies to:

- All custom integration tasks triggered from Icertis workflows or agreement events (Publish, Execute, Amend, etc.).
- All retry and logging mechanisms related to external API communication.
- Implementations performed by internal or partner teams on the Icertis platform.

It does not cover product-level configurations, UI customizations, or core Icertis upgrades.

Objectives

- Define standard design principles for building reliable and reusable integration tasks.
- Ensure clear separation of logic between task layers, helper methods, and event handlers.
- Establish fault-tolerant retry and logging frameworks for predictable behavior during system or API failures.
- Encourage consistent development practices across projects for maintainability and easier audits.
- Provide a reference for new developers and solution architects joining ongoing Icertis implementations.

1. Failure Handling

- **Internal Exceptions**
  - When a task fails due to internal exceptions within the Icertis system, mark the task as IsSuccessful = 0 .
  - This lets the Icertis task framework automatically retry the operation based on its built-in retry logic.
- **External API Failures**
  - For failures caused by third-party APIs, implement a custom **exponential back-off retry mechanism** .
  - Include progressive backoff intervals and a maximum retry threshold to prevent overload or throttling.

2. Task Design Principles

- **Helper Method Pattern**
  - Encapsulate all logic inside helper methods rather than directly inside task methods.
  - This ensures reusability—other tasks or statuses can call the same business logic without duplication.
- **Event-Based Task Separation**
  - Every **Publish** event for an agreement should have its own dedicated custom task.

    - Every **Executed** event for an agreement should also have a separate task.
    - Avoid combining event logic within a single task unless necessary.

- **Shared Logic Across Events**
    - If one integration serves both Publish and Executed events, create **two distinct custom integration tasks** that both call a **shared helper method** for the common logic.

- **Multiple Integrations on Publish**
    - If multiple integrations trigger on the same Publish event, use the **Chain of Responsibility** pattern.
    - Each integration should execute sequentially, and the final state of the Agreement object should be persisted only **after** all integrations complete successfully.

3. Logging Strategy

Centralized Master Data for Logs

- Maintain a master data table for logging
- This approach aligns with current Icertis best practices and newer implementations.
- Include metadata such as:
    - Integration name
    - Task ID
    - Execution time
    - Status (Success/Failure)
    - Error message or exception trace
    - Retry count

4. Recommended Implementation Standards

Code Organization

- Keep helper methods under a consistent namespace or class (e.g., IntegrationHelper or TaskUtils).
- Ensure every integration task has its own configuration key or metadata entry for easy toggling and monitoring.

Testing

- Validate retry mechanisms using mock failures in both internal (Icertis API) and external (3rd-party API) contexts.
- Use sandbox API endpoints wherever available.

Error Propagation

- Only raise exceptions after retries are exhausted.
- When logging, include stack trace and request/response payloads (masking any sensitive data).

5. Summary

Area	Best Practice	Purpose
Internal Failures	IsSuccessful = 0	Enables auto-retry by Icertis
3rd-Party Failures	Exponential retry	Prevents API throttling
Code Placement	Helper methods	Promotes reuse and clarity
Task Setup	Separate per event	Ensures clean responsibility
Multiple Integrations	Chain of responsibility	Maintains sequence and integrity
Logging	Master data-based	Simplifies traceability