RevenueOS: The Implementation Series

ROS-02 · Module 2 · FOUNDATION

Contract Lifecycle Management

Contract authoring, negotiation, execution, obligation tracking, amendment management, renewal triggers, and revenue-connected contract intelligence

If it is not in the contract data model, the billing system will not honour it.

Vendor-agnostic specification · For all vendor comparisons see ROS-23

PREFACE

The Contract Governs Everything. This Module Specifies How to Govern the Contract.

The commercial foundation before the first customer is acquired.

Every AI company will eventually face the same question from a customer: "We agreed to X — why are you billing us for Y?" The answer lives in the contract. The quality of the answer — how quickly it can be produced, how precisely it maps billing events to contract terms, how defensible it is in a dispute — depends entirely on how well the contract was captured in the commercial data model at signature.

Contract lifecycle management in the context of revenue operations is not primarily about legal workflow. Legal teams care about clause libraries, redline collaboration, and

risk scoring. Revenue operations teams care about something more specific: whether the commercial terms that determine billing, entitlements, pricing, and recognition are captured as structured data that downstream systems can act on automatically.

A contract that lives as a PDF in a document repository is commercially inert. A contract whose terms have been extracted into the canonical objects defined in this book — pricing plans, entitlement rules, SLA commitments, payment terms, renewal triggers — is commercially active. It governs billing automatically. It fires entitlement events. It alerts the renewals team 90 days before expiry. It creates the RevRec performance obligation schedule on signature.

This book specifies the complete CLM architecture from both angles: the workflow angle (how contracts are authored, negotiated, approved, and executed) and the data angle (how contract terms are extracted, validated, and connected to downstream commercial systems). Both are required. A CLM with great workflow and weak extraction produces beautifully executed contracts that the billing system cannot read. A CLM with great extraction and weak workflow produces structured data from contracts that were never properly approved.

Three principles govern this book. Structure before storage: a contract that is stored as a document is less valuable than a contract whose commercial terms are stored as structured data. Extraction is not optional: manual re-entry of contract terms into billing and entitlement systems is the primary source of configuration errors and revenue leakage. Contracts govern billing, not the reverse: when the billing configuration and the contract disagree, the contract is correct and the billing configuration must be corrected.

CHAPTER 2.1

CLM Architecture: Contract Objects, Lifecycle States, and Commercial Extraction

Learning Objectives

• Define the canonical contract object schema and identify which fields connect directly to downstream billing and entitlement systems

• Specify the complete contract state machine including all states, valid transitions, and the commercial events triggered at each transition

• Design the commercial term extraction pipeline that maps contract text to structured canonical objects

• Establish extraction confidence thresholds and the exception handling workflow for low-confidence extractions

• Understand the relationship between the contract object and the downstream objects it governs: PricingPlan, Entitlement, RevRec schedule, and payment terms

Deployment Context Guidance

AI-NATIVE SaaS	TRANSITIONING ENTERPRISE	LARGE INTERNAL PLATFORM
Start with a simple extraction schema covering the 12 most commercially critical fields: customer, product, price, billing period, start date, end date, auto-renew flag, SLA tier, payment terms, discount, governing law, and primary contact. Add fields as you discover what your billing system needs — over-engineering the extraction schema before you understand your billing edge cases creates maintenance burden without value.	The most common CLM migration failure: extracting contract terms into a new system without first auditing whether the terms in legacy contracts are consistent with current billing configurations. Run a reconciliation between your legacy contract repository and your billing system before migration. The gaps you find are the leakage you have already incurred.	Large internal platforms typically have the most complex contract structures: multi-entity, multi-currency, multi-product, with custom SLAs and bespoke payment terms negotiated over many years. The extraction schema must accommodate this complexity without requiring a different extraction path for every customer. Design for the most complex case first — simple contracts are a subset of complex contracts.

Process Flow

1. Contract object creation: when a contract is initiated — either through the deal desk (ROS-04) or directly in the CLM — a Contract object is created in DRAFT status. At this stage the object captures the parties, the proposed products, the pricing references, and the expected terms.
2. Contract authoring: the contract document is authored using templates from the clause library. Each template maps to a set of extracted fields — the system knows which text sections correspond to which canonical fields.
3. Negotiation and redlines: the contract document is shared with the counterparty. All changes are tracked as redlines. Each redline that affects a commercial term triggers a re-extraction of the affected fields.
4. Internal approval: the contract routes through the approval chain based on deal value, non-standard term flags, and risk score. Approval is recorded as an Approval object linked to the Contract.
5. Execution: the contract is signed via e-signature integration. On signature, the Contract transitions to EXECUTED status. Execution triggers the extraction pipeline.
6. Extraction: the extraction engine processes the executed contract, mapping text to canonical fields and generating a confidence score for each. Fields below the confidence threshold are flagged for human review. High-confidence fields are auto-populated into downstream objects.
7. Downstream activation: when extraction is complete and reviewed, the system creates or updates: the Account billing configuration, the Entitlement objects, the RevRec performance obligation schedule, and the renewal trigger. The Contract is now commercially active.
8. Performance monitoring: throughout the contract term, the system monitors SLA performance, tracks obligation milestones, and monitors renewal trigger dates. Alerts fire when monitoring events occur.
9. Renewal or expiry: at the renewal trigger date (typically 90 days before expiry), the renewal workflow is initiated. If not renewed, the contract expires and entitlements are deactivated.

Data Model

```
OBJECT: Contract
  id                        UUID          REQUIRED
  account_id                UUID          REQUIRED    FK → Account.id
  contract_number           string        REQUIRED    Human-readable reference;
unique; system-generated or manual
  contract_type             enum          REQUIRED    MSA | ORDER_FORM |
AMENDMENT | ADDENDUM | SOW | NDA | RENEWAL
  status                    enum          REQUIRED    DRAFT | IN_REVIEW |
APPROVED | SENT_FOR_SIGNATURE | EXECUTED | EXPIRED | TERMINATED | SUPERSEDED
  parent_contract_id        UUID          OPTIONAL    FK → Contract.id; for
amendments and addenda
  primary_owner_user_id     UUID          REQUIRED    Commercial owner
(Sales/RevOps)
  legal_owner_user_id       UUID          REQUIRED    Legal owner (Legal team)
  counterparty_name         string        REQUIRED    Legal entity name of the
customer
```

```
  counterparty_signing_authority string   OPTIONAL    Name and title of
counterparty signatory

  -- Commercial Terms (extracted) --
  product_ids               UUID[]        REQUIRED    FK[] → Product.id; products
covered by this contract
  pricing_plan_ids          UUID[]        REQUIRED    FK[] → PricingPlan.id;
pricing plans in effect
  total_contract_value_cents integer      OPTIONAL    Total committed value over
contract term (cents)
  arr_cents                 integer       OPTIONAL    Annualised recurring
revenue component (cents)
  start_date                ISO 8601 date REQUIRED
  end_date                  ISO 8601 date OPTIONAL    null = evergreen contract
  billing_period            enum          REQUIRED    MONTHLY | QUARTERLY |
ANNUAL | CUSTOM
  payment_terms_days        integer       REQUIRED    e.g., 30 for Net-30; 0 for
payment-on-invoice
  payment_method            enum          OPTIONAL    CREDIT_CARD | ACH | WIRE |
PO | CHECK
  po_number                 string        OPTIONAL    Purchase order number if
applicable
  currency                  ISO 4217      REQUIRED

  -- Auto-renewal --
  auto_renew                boolean       REQUIRED    default false
  auto_renew_notice_days    integer       CONDITIONAL Required when auto_renew =
true; notice period to cancel
  renewal_term_months       integer       OPTIONAL    Length of auto-renewal
term; null = same as original term

  -- SLA --
  sla_tier                  string        OPTIONAL    Reference to SLA tier
definition
  sla_uptime_commitment     decimal       OPTIONAL    e.g., 0.999 = 99.9% uptime
  sla_response_time_hours   integer       OPTIONAL    First response SLA for
support tickets (hours)
  sla_credit_formula        string        OPTIONAL    Human-readable description
of SLA credit calculation

  -- Governance --
  governing_law             string        OPTIONAL    e.g., "California, United
States"
  dispute_resolution        enum          OPTIONAL    ARBITRATION | LITIGATION |
MEDIATION
  liability_cap_cents       integer       OPTIONAL    Maximum liability cap
(cents)
  data_processing_agreement boolean       REQUIRED    default false; true if DPA
has been executed
  hipaa_baa                 boolean       REQUIRED    default false
  federal_addendum          boolean       REQUIRED    default false

  -- Extraction metadata --
  extraction_status         enum          REQUIRED    NOT_EXTRACTED | IN_PROGRESS
| NEEDS_REVIEW | COMPLETE | FAILED
  extraction_confidence     decimal       OPTIONAL    0.00–1.00; overall
confidence score from extraction engine
```

```
  extraction_completed_at    ISO 8601 UTC  OPTIONAL
  extraction_reviewed_by     UUID          OPTIONAL    User who reviewed
extraction results

  -- Lifecycle --
  created_at                 ISO 8601 UTC  SYSTEM
  executed_at                ISO 8601 UTC  OPTIONAL    Set on EXECUTED transition
  expires_at                 ISO 8601 UTC  OPTIONAL    Set from end_date; used for
renewal trigger calculation
  terminated_at              ISO 8601 UTC  OPTIONAL
  termination_reason         string        OPTIONAL

OBJECT: ContractExtractionField
  id                         UUID          REQUIRED
  contract_id                UUID          REQUIRED    FK → Contract.id
  field_name                 string        REQUIRED    Name of the canonical field
being extracted
  extracted_value            string        REQUIRED    Raw extracted value (text)
  mapped_value               string        OPTIONAL    Value mapped to canonical
format (e.g., "Net 30" → 30)
  confidence_score           decimal       REQUIRED    0.00–1.00
  source_text                string        OPTIONAL    The verbatim contract text
from which this was extracted
  source_page                integer       OPTIONAL
  source_section             string        OPTIONAL    e.g., "Section 4.2 —
Payment Terms"
  review_status              enum          REQUIRED    AUTO_ACCEPTED |
PENDING_REVIEW | HUMAN_CONFIRMED | HUMAN_CORRECTED | REJECTED
  reviewed_by                UUID          OPTIONAL
  reviewed_at                ISO 8601 UTC  OPTIONAL
  correction_notes           string        OPTIONAL
```

BUSINESS RULES

CLM-001: No Contract may transition to EXECUTED status without both primary_owner_user_id and legal_owner_user_id being assigned.

CLM-002: No billing configuration may be created or modified based on a Contract in DRAFT or IN_REVIEW status. Only EXECUTED contracts may trigger downstream commercial object creation.

CLM-003: Extraction confidence threshold for auto-acceptance is 0.85 on all commercial fields (start_date, end_date, billing_period, payment_terms_days, arr_cents). Fields below 0.85 must be reviewed by the primary commercial owner before downstream activation occurs.

CLM-004: A Contract with extraction_status = NEEDS_REVIEW may not trigger downstream activation until extraction_status transitions to COMPLETE. The system must enforce this gate.

CLM-005: parent_contract_id must reference a Contract in EXECUTED status. An amendment to a DRAFT or IN_REVIEW contract is not permitted — amendments can only modify executed agreements.

CLM-006: When a Contract is TERMINATED, all Entitlement objects linked to that contract must be deactivated within one billing cycle. Active entitlements on a terminated contract are a compliance finding.

CLM-007: auto_renew = true requires auto_renew_notice_days to be populated. The renewal notice workflow must fire at (end_date minus auto_renew_notice_days) to give the customer the required notice

window to cancel.

CLM-008: hipaa_baa = true requires a corresponding BAA document reference in the contract document repository. hipaa_baa = true without a BAA document is a HIPAA compliance violation.

API Specification

```
POST   /contracts
       Auth: CONTRACT_WRITE scope
       Request: account_id, contract_type, counterparty_name, start_date,
product_ids, pricing_plan_ids, billing_period, payment_terms_days, currency,
primary_owner_user_id, legal_owner_user_id, [all optional commercial fields]
       Response: 201 — Contract object in DRAFT status
       Errors: 400 (missing required field, invalid product_id, invalid
pricing_plan_id), 403

GET    /contracts/{id}
       Auth: CONTRACT_READ
       Response: 200 — Contract object with embedded extraction status
       Errors: 404, 403

PATCH  /contracts/{id}
       Auth: CONTRACT_WRITE
       Request: Any mutable field (not id, contract_number, executed_at,
created_at)
       Response: 200 — updated Contract
       Errors: 400, 403, 409 (cannot modify EXECUTED contract without amendment)

POST   /contracts/{id}/submit-for-approval
       Auth: CONTRACT_WRITE
       Triggers: Routes to approval chain based on contract value and non-
standard term flags
       Response: 200 — Contract in IN_REVIEW status + ApprovalRequest array
       Errors: 400 (missing required commercial fields), 403, 409 (already in
approval)

POST   /contracts/{id}/execute
       Auth: CONTRACT_EXECUTE scope (restricted)
       Request: { signature_provider, signature_reference_id, executed_at }
       Response: 200 — Contract in EXECUTED status; triggers extraction pipeline
       Errors: 400 (not in APPROVED status), 403

POST   /contracts/{id}/extract
       Auth: CONTRACT_WRITE (or triggered automatically on execution)
       Response: 202 Accepted — extraction job queued
       Errors: 400 (contract not EXECUTED), 403

GET    /contracts/{id}/extraction-status
       Auth: CONTRACT_READ
       Response: 200 — { extraction_status, confidence_score, fields:
[ContractExtractionField], fields_needing_review: count }
       Errors: 404, 403
```

```
POST   /contracts/{id}/extraction-fields/{field_id}/review
       Auth: CONTRACT_WRITE
       Request: { review_status, mapped_value (if corrected), correction_notes }
       Response: 200 — updated ContractExtractionField
       Errors: 400, 403, 404

POST   /contracts/{id}/activate-downstream
       Auth: CONTRACT_EXECUTE
       Pre-condition: extraction_status = COMPLETE
       Triggers: Creates/updates Account billing config, Entitlements, RevRec
schedule, renewal trigger
       Response: 200 — { billing_config_id, entitlement_ids[],
revrec_schedule_id, renewal_trigger_id }
       Errors: 400 (extraction not complete), 403, 422 (downstream creation
failed — details in response)

GET    /contracts
       Auth: CONTRACT_READ
       Query: account_id, status, expiring_within_days, auto_renew,
extraction_status, page, page_size
       Response: 200 — paginated Contract list
       Errors: 400
```

Exception Handling and Edge Cases

1. Extraction confidence below threshold on a critical commercial field

The extraction engine returns confidence 0.71 on the payment_terms_days field — below the 0.85 threshold. The system sets extraction_status = NEEDS_REVIEW and creates a review task for the primary commercial owner. The downstream activation gate is blocked until the field is reviewed. The reviewer inspects the source text, confirms the value (or corrects it), and marks the field as HUMAN_CONFIRMED. When all fields are at HUMAN_CONFIRMED or AUTO_ACCEPTED status, extraction_status transitions to COMPLETE and downstream activation can proceed.

The failure mode to prevent: a field that was extracted with low confidence and not reviewed before activation. This creates a billing configuration that may not match the contract — the most common source of billing disputes in CLM-integrated systems.

2. Contract contains contradictory terms

During extraction, the engine identifies two conflicting payment terms: Section 3.2 states "Net 30" and Section 8.1 (an addendum incorporated by reference) states "Net 45." The engine cannot resolve the conflict automatically. The field is flagged with review_status = PENDING_REVIEW and

confidence_score = 0.00. The legal owner must review the conflict, determine which section governs (typically the addendum takes precedence over the base contract), and HUMAN_CONFIRM the correct value. The resolution is documented in correction_notes for audit purposes.

3. Addendum changes a term in the base contract

A signed addendum changes the pricing from the original MSA. The addendum is created as a new Contract with contract_type = ADDENDUM and parent_contract_id referencing the original MSA. When both are EXECUTED, the system must determine which term governs. Rule: ADDENDUM terms supersede parent contract terms for fields that are explicitly addressed in the addendum. The downstream billing configuration is updated to reflect the addendum terms. The original MSA's pricing plan is not removed — it is superseded, and the supersession is recorded in the audit trail.

4. Execution of contract with future start date

A contract is signed on April 15 with a start_date of May 1. The Contract transitions to EXECUTED on April 15. Downstream activation creates the Entitlement and RevRec schedule objects but sets them to PENDING status until May 1. Billing does not begin until the start_date. The billing engine's pre-activation check must query start_date, not execution date. Missing this check results in early billing — a billing dispute that requires a credit.

5. Multi-entity contract with different billing terms per entity

A global enterprise signs a single MSA covering three subsidiaries with different billing currencies and payment terms (US entity: USD, Net-30; EU entity: EUR, Net-45; APAC entity: SGD, Net-60). The Contract object has a single parent record but three ContractBillingEntity child objects, each with entity-specific billing terms. The extraction engine must identify and separate the per-entity terms — this requires the extraction schema to support multi-entity contract structures.

Integration Checklist

- □ POST /contracts creates contract in DRAFT status with all required fields validated
- □ CLM-003 (confidence threshold 0.85) is enforced — fields below threshold blocked from auto-acceptance
- □ CLM-004 (downstream activation gate) is enforced — NEEDS_REVIEW contracts cannot trigger billing config
- □ POST /contracts/{id}/execute requires APPROVED status — execution of unapproved contracts returns 409

- ☐ Extraction pipeline fires automatically on EXECUTED transition
- ☐ ContractExtractionField records are created for all extractable fields with confidence scores
- ☐ Review task created and assigned to primary_owner_user_id for all fields with confidence < 0.85
- ☐ POST /contracts/{id}/activate-downstream creates Entitlement objects with correct start_date
- ☐ Entitlements created from contracts with future start_date have status PENDING until start_date
- ☐ RevRec performance obligation schedule created on activation with correct recognition pattern
- ☐ Renewal trigger fires at (end_date minus auto_renew_notice_days) when auto_renew = true
- ☐ Contract TERMINATION deactivates all linked Entitlements within one billing cycle
- ☐ Audit log captures every status transition with actor, timestamp, and before/after state
- ☐ hipaa_baa = true is validated against presence of BAA document reference
- ☐ Multi-entity contracts create separate billing configurations per entity

CHAPTER 2.1 — Key Takeaways
› The Contract object is the commercial source of truth — extraction connects it to every downstream billing and entitlement object
› The 0.85 confidence threshold gate (CLM-003 and CLM-004) is the most important control in this chapter — it prevents mis-extracted terms from propagating to billing
› Contract state machine has 9 states — only EXECUTED contracts may trigger downstream commercial objects
› Multi-entity contracts require per-entity billing configurations — the parent contract is the legal anchor, not the billing authority
› The contract.executed event on the event bus is the integration contract for all downstream systems — they subscribe to this event, not to polling the CLM

CHAPTER 2.2

Contract Authoring: Templates, Clause Libraries, and Guided Assembly

Learning Objectives

• Design a template architecture that speeds authoring while enforcing pre-approved commercial structures

• Build a clause library that separates standard, negotiable, and non-negotiable clauses with clear governance

• Specify the guided assembly workflow that helps deal desk teams construct contracts without legal involvement for standard deals

• Configure the non-standard clause detection rules that route contracts to legal review automatically

• Define the template version management process that keeps templates current without disrupting in-flight contracts

Process Flow

1. Template selection: the deal desk or legal team selects a contract template based on deal type (new business, renewal, expansion, partner) and customer segment (SMB, mid-market, enterprise, regulated). Template selection may be automated based on deal attributes from the CRM.
2. Variable population: the system populates contract variables (customer name, product names, pricing, start date, SLA tier) from the deal record. Variables are validated — pricing variables must reference active PricingPlans; date variables must be consistent.
3. Optional clause selection: the deal desk selects from approved optional clauses. Each optional clause is tagged with the conditions under which it is available (deal value threshold, customer segment, geographic jurisdiction). Optional clauses not meeting the conditions are not displayed.
4. Non-standard term detection: if the deal desk attempts to add text outside the clause library, or modifies standard clause text beyond approved deviations, the system flags the modification as a non-standard term and routes to legal review.
5. Internal consistency check: before the contract is sent for negotiation, an automated check validates internal consistency — that the pricing variables match the referenced PricingPlan, that the term dates are consistent, that SLA commitments reference defined SLA tiers.
6. Document generation: the contract document is generated from the template and variable values, producing a clean PDF and an editable version for negotiation.

Data Model

```
OBJECT: ContractTemplate
  id                    UUID          REQUIRED
  name                  string        REQUIRED
  contract_type         enum          REQUIRED    MSA | ORDER_FORM | AMENDMENT |
SOW | RENEWAL
  customer_segments     enum[]        REQUIRED    SMB | MID_MARKET | ENTERPRISE |
REGULATED | PARTNER
  version               string        REQUIRED    e.g., "2025.Q2.3"
  status                enum          REQUIRED    DRAFT | ACTIVE | DEPRECATED
  effective_from        ISO 8601 date REQUIRED
```

```
  effective_to          ISO 8601 date OPTIONAL
  legal_approved_by     UUID          REQUIRED
  legal_approved_at     ISO 8601 UTC  REQUIRED
  template_document_url string        REQUIRED    URL to the template document in
the document store
  variables             TemplateVariable[]  REQUIRED
  required_clauses      ClauseRef[]   REQUIRED    Clauses that must appear in
every contract using this template
  optional_clauses      ClauseRef[]   OPTIONAL    Clauses available for selection
  non_negotiable_clause_ids UUID[]    REQUIRED    Clause IDs that cannot be
modified

OBJECT: Clause
  id                    UUID          REQUIRED
  name                  string        REQUIRED
  clause_type           enum          REQUIRED    STANDARD | OPTIONAL |
NON_NEGOTIABLE | JURISDICTION_SPECIFIC
  topic                 enum          REQUIRED    PAYMENT | LIABILITY | IP |
DATA_PRIVACY | SLA | TERMINATION | GOVERNING_LAW | CONFIDENTIALITY | OTHER
  clause_text           string        REQUIRED    The approved clause text (may
contain variable placeholders)
  approved_deviations   string[]      OPTIONAL    Pre-approved alternative
wordings that do not require legal review
  risk_score            integer       REQUIRED    1–5 (1 = low risk, 5 = high
risk); used for approval routing
  jurisdiction          string        OPTIONAL    null = universal; otherwise
jurisdiction-specific clause
  version               string        REQUIRED
  legal_approved_by     UUID          REQUIRED
  last_updated_at       ISO 8601 UTC  SYSTEM

OBJECT: TemplateVariable
  variable_name         string        REQUIRED    e.g.,
"{{customer_legal_name}}", "{{arr_value}}"
  source                enum          REQUIRED    CRM_FIELD | DEAL_RECORD |
CATALOG | USER_INPUT
  source_field          string        CONDITIONAL Required when source ≠
USER_INPUT
  validation_rule       string        OPTIONAL    Regex or reference to a named
validation function
  required              boolean       REQUIRED    default true
```

BUSINESS RULES

TMPL-001: A ContractTemplate may not be marked ACTIVE without legal_approved_by and legal_approved_at being set. Legal approval is mandatory — no exceptions.

TMPL-002: non_negotiable_clause_ids must be reviewed and re-confirmed by legal every 12 months. Templates with non-negotiable clauses not reviewed in the past 12 months are automatically flagged for legal review.

TMPL-003: Template variables sourced from CRM_FIELD or DEAL_RECORD are auto-populated and locked after population. They cannot be manually overridden without a non-standard term flag.

TMPL-004: Any modification to a NON_NEGOTIABLE clause text during negotiation automatically

escalates to General Counsel review regardless of deal value or standard approval chain.

TMPL-005: Template versions are forward-only. A template in ACTIVE status cannot be rolled back — corrections require a new version. The old version is DEPRECATED, not deleted.

TMPL-006: Contracts in-flight (DRAFT or IN_REVIEW) against a template that is subsequently DEPRECATED continue to use the deprecated template version. They are not automatically updated to the new version.

API Specification

```
GET    /clm/templates
       Auth: CONTRACT_READ
       Query: contract_type, customer_segment, status
       Response: 200 — paginated ContractTemplate list

POST   /clm/templates/{id}/generate-document
       Auth: CONTRACT_WRITE
       Request: { contract_id, variable_values: {}, selected_optional_clause_ids:
[] }
       Response: 202 Accepted — { job_id, estimated_completion_seconds }
       Async result: document_url, internal_consistency_check_result,
non_standard_flags[]

GET    /clm/clauses
       Auth: CONTRACT_READ
       Query: clause_type, topic, jurisdiction, status
       Response: 200 — paginated Clause list

POST   /clm/contracts/{id}/non-standard-terms
       Auth: CONTRACT_WRITE
       Request: { clause_id_or_section, proposed_text, business_justification }
       Response: 201 — NonStandardTerm object; routes to legal review queue
       Errors: 400 (no justification), 403
```

CHAPTER 2.2 — Key Takeaways

› Templates are the primary control preventing non-standard commercial structures from entering the pipeline without legal review

› The clause library separates standard, optional, and non-negotiable clauses — each category has different governance rules

› TMPL-001 (legal approval required) is non-negotiable — a template that was not legally approved is a liability

› Template variables sourced from CRM or deal records are auto-populated and locked — preventing manual override errors

› TMPL-006 (in-flight contracts not updated when template is deprecated) prevents mid-contract

changes to template definitions

CHAPTER 2.3

Negotiation Workflow: Redlines, Version Control, and Collaboration

Learning Objectives

• Design the redline management workflow that maintains a complete version history of all contract changes

• Specify the commercial term impact assessment triggered by any redline affecting a billable term

• Configure the collaboration model for internal and external contract review with appropriate access controls

• Define the redline approval rules that determine which changes require legal sign-off before acceptance

• Build the audit trail for negotiation history that documents who proposed and who accepted each change

Process Flow

1. Contract sent to counterparty: the approved draft is shared with the customer via a secure link or exported to Word. The system records the send event with timestamp and document version.

2. Counterparty redlines received: the customer returns the redlined document. The system receives the document and parses the redlines into individual change records.

3. Redline classification: each redline is classified by type (addition, deletion, modification) and topic (payment terms, liability, IP, SLA, other). Redlines affecting commercial terms (pricing, payment, SLA, term dates) are flagged with higher priority.

4. Internal review routing: the classified redlines are routed to the appropriate reviewers. Commercial term redlines route to the primary commercial owner and deal desk. Legal term redlines route to the legal owner. High-risk clause modifications (risk_score ≥ 4) route to General Counsel.

5. Accept/reject/counter: reviewers accept, reject, or propose counter-language for each redline. Accepted changes are incorporated into the contract version. Counter-proposals trigger a new document version and are shared with the counterparty.

6. Version tracking: every document version is stored with a complete diff from the previous version. The negotiation history shows who proposed each change and when it was accepted or rejected.

7. Commercial impact tracking: when a redline affecting a commercial term is accepted, the system updates the pending extraction fields for that term. The deal desk is notified to review the impact on the deal record and RevRec schedule.

Data Model

```
OBJECT: ContractVersion
  id                    UUID          REQUIRED
  contract_id           UUID          REQUIRED
  version_number        integer       REQUIRED    Sequential; 1 = first draft
  document_url          string        REQUIRED    URL to the version document in
document store
  created_by            UUID          REQUIRED    SYSTEM for auto-generated;
user_id for manually created
  created_at            ISO 8601 UTC  SYSTEM
  change_summary        string        OPTIONAL    Human-readable summary of
changes from previous version
  diff_from_previous    object        SYSTEM      Structured diff (additions[],
deletions[], modifications[])
  sent_to_counterparty  boolean       REQUIRED    default false
  sent_at               ISO 8601 UTC  OPTIONAL

OBJECT: Redline
  id                    UUID          REQUIRED
  contract_version_id   UUID          REQUIRED
  change_type           enum          REQUIRED    ADDITION | DELETION |
MODIFICATION
  topic                 enum          REQUIRED    (same as Clause.topic enum)
  is_commercial_term    boolean       REQUIRED    Does this change affect a
billable/extractable field?
  affected_field        string        OPTIONAL    The canonical field name if
is_commercial_term = true
  original_text         string        OPTIONAL    null for ADDITION
  proposed_text         string        OPTIONAL    null for DELETION
  proposed_by           enum          REQUIRED    INTERNAL | COUNTERPARTY
  proposed_by_user_id   UUID          OPTIONAL    Null when proposed_by =
COUNTERPARTY
  status                enum          REQUIRED    PENDING | ACCEPTED | REJECTED |
COUNTERED
  decided_by            UUID          OPTIONAL
  decided_at            ISO 8601 UTC  OPTIONAL
  counter_redline_id    UUID          OPTIONAL    FK → Redline.id when status =
COUNTERED
```

BUSINESS RULES

REDLINE-001: A Redline with is_commercial_term = true that is ACCEPTED must trigger an update to the corresponding ContractExtractionField with review_status = NEEDS_REVIEW. The extraction system cannot auto-accept values from accepted redlines — human confirmation is required.

REDLINE-002: Redlines modifying clauses with risk_score = 5 cannot be ACCEPTED by anyone below General Counsel in the approval hierarchy.

REDLINE-003: All Redlines on non-negotiable clauses must be REJECTED. The system enforces this — non-negotiable clause redlines from the counterparty are automatically REJECTED with a standard response explaining they cannot be accepted.

REDLINE-004: A contract may not transition to APPROVED status with any Redlines in PENDING status. All redlines must be ACCEPTED, REJECTED, or COUNTERED before approval routing begins.

REDLINE-005: The negotiation history (all ContractVersions and Redlines) is immutable. Documents and change records cannot be deleted, only superseded. This preserves the complete negotiation audit trail.

API Specification

```
POST   /contracts/{id}/versions
       Auth: CONTRACT_WRITE
       Request: { document_url, change_summary }
       Response: 201 — ContractVersion; triggers redline parsing job
       Errors: 400, 403

GET    /contracts/{id}/versions
       Auth: CONTRACT_READ
       Response: 200 — array of ContractVersion objects in version_number order

GET    /contracts/{id}/versions/{version_id}/redlines
       Auth: CONTRACT_READ
       Query: status, is_commercial_term, topic
       Response: 200 — filtered Redline list

PATCH  /contracts/{id}/redlines/{redline_id}
       Auth: CONTRACT_WRITE
       Request: { status (ACCEPTED | REJECTED | COUNTERED), counter_redline_id
(if COUNTERED), decided_by }
       Response: 200 — updated Redline; if ACCEPTED commercial term, creates
ContractExtractionField review task
       Errors: 400 (ACCEPTED on non-negotiable), 403, 404
```

CHAPTER 2.3 — Key Takeaways

› Every redline is a structured record — not just a track-change in a Word document

› REDLINE-001: accepted commercial term redlines must be reviewed by the commercial owner before downstream activation — never auto-accepted

› REDLINE-003: non-negotiable clause redlines from counterparty are automatically rejected — the system enforces this, not a human

› REDLINE-004: no contract proceeds to approval with PENDING redlines — every redline must be resolved before approval routing

› Negotiation history is immutable — the complete redline trail from first draft to execution is the

audit record for the deal

CHAPTER 2.4

Approval Workflows: Multi-Tier, Conditional, and Parallel Approvals

Learning Objectives

• Design a contract approval authority matrix that routes contracts to the correct approvers based on deal value, non-standard terms, and risk score

• Specify the conditional approval rules that add approval steps when specific contract attributes are present

• Configure parallel approval tracks for legal and commercial review that proceed simultaneously without blocking each other

• Define SLA requirements for each approval level and the escalation logic when SLAs are breached

• Build the approval audit trail that documents the complete decision chain for every executed contract

Process Flow

1. Approval routing: when the contract owner submits the contract for approval, the routing engine evaluates the contract attributes against the approval authority matrix. The engine determines the required approval chain.

2. Parallel track initiation: legal review and commercial review proceed in parallel. Commercial track: deal desk → Sales Manager → VP Sales (if value threshold). Legal track: legal counsel → General Counsel (if non-standard terms above risk threshold). Finance track (if RevRec flag): RevRec Analyst → Controller.

3. SLA monitoring: each approval step has an SLA. If the SLA is breached without a decision, the system escalates to the next authority level and notifies both the approver and their manager.

4. Conditional steps: certain contract attributes trigger additional approval steps. hipaa_baa = true adds a compliance officer step. liability_cap below the company standard requires CFO approval. Federal contracts require security review.

5. Approval or rejection: each approver approves, rejects, or requests changes. Rejection routes the contract back to the owner with documented rejection reasons. Change requests add the requested changes to the next contract version.

6. Final approval and execution clearance: when all tracks are approved, the contract transitions to APPROVED status and is cleared for execution (signature collection).

Data Model

```
OBJECT: ContractApproval
  id                    UUID          REQUIRED
  contract_id           UUID          REQUIRED
  approval_track        enum          REQUIRED    COMMERCIAL | LEGAL | FINANCE |
COMPLIANCE | SECURITY
  sequence_order        integer       REQUIRED    Order within the track (1 =
first)
  approver_role         string        REQUIRED    Required role for this approval
step
  approver_user_id      UUID          OPTIONAL    Specific approver (null = any
user with required role)
  status                enum          REQUIRED    PENDING | APPROVED | REJECTED |
CHANGES_REQUESTED | DELEGATED | SKIPPED
  sla_hours             integer       REQUIRED    Hours within which this
approval must be completed
  sla_deadline          ISO 8601 UTC  SYSTEM      = submitted_at + sla_hours
  decided_at            ISO 8601 UTC  OPTIONAL
  decision_notes        string        OPTIONAL    Required when status = REJECTED
or CHANGES_REQUESTED
  delegated_to          UUID          OPTIONAL    FK → User.id when status =
DELEGATED
  escalation_count      integer       SYSTEM      Number of times this step has
escalated

OBJECT: ContractApprovalMatrix
  deal_value_floor_cents      integer   REQUIRED    Minimum deal value this rule
applies to
  deal_value_ceiling_cents    integer   OPTIONAL    Maximum deal value; null = no
upper limit
  required_commercial_approvers string[] REQUIRED  List of roles in order
  required_legal_approvers    string[]  REQUIRED
  required_finance_approvers  string[]  OPTIONAL
  conditional_triggers        ApprovalTrigger[] OPTIONAL

OBJECT: ApprovalTrigger
  condition_field       string        REQUIRED    Contract field that triggers
the condition
  condition_operator    enum          REQUIRED    EQUALS | LESS_THAN |
GREATER_THAN | IS_TRUE | IS_FALSE | CONTAINS
  condition_value       string        REQUIRED
  additional_approver_role string     REQUIRED    Role added to approval chain
when condition is met
  approval_track        enum          REQUIRED    Which track this additional
approver belongs to
```

BUSINESS RULES

APPR-001: A contract may not proceed to execution without at least one COMMERCIAL approval and one

LEGAL approval recorded.
APPR-002: REJECTION of a contract by any approver on any track cancels all pending approvals across all tracks. The contract returns to DRAFT status with the rejection reason documented.
APPR-003: Delegations may only be created by the delegating approver while they are active in the system. Expired users cannot create delegations retroactively.
APPR-004: SLA breach escalation: if an approver does not decide within their SLA, the step is automatically escalated to the next authority level. The original approver loses the active approval step but retains visibility. Escalation is logged.
APPR-005: conditional_triggers are evaluated at the time of approval routing. A change to the contract that would modify a conditional trigger field after routing has begun requires re-routing.
APPR-006: ContractApproval.decision_notes is mandatory when status = REJECTED or CHANGES_REQUESTED. A rejection without documented reasons is non-compliant.

API Specification

```
POST   /contracts/{id}/submit-for-approval
       Auth: CONTRACT_WRITE
       Pre-condition: Contract in DRAFT, all required fields populated, no
PENDING redlines
       Response: 200 — Contract in IN_REVIEW, array of ContractApproval objects
created
       Errors: 400 (pending redlines, missing fields), 403, 409 (already in
approval)

GET    /contracts/{id}/approvals
       Auth: CONTRACT_READ
       Response: 200 — all ContractApproval objects grouped by track

POST   /contracts/{id}/approvals/{approval_id}/decide
       Auth: Requires role matching ContractApproval.approver_role
       Request: { status, decision_notes (required for
REJECTED/CHANGES_REQUESTED) }
       Response: 200 — updated ContractApproval; if all tracks complete, Contract
transitions to APPROVED
       Errors: 400 (missing notes on rejection), 403 (wrong role), 404

POST   /contracts/{id}/approvals/{approval_id}/delegate
       Auth: Current active approver for this step
       Request: { delegated_to_user_id, reason }
       Response: 200 — updated ContractApproval with status DELEGATED
       Errors: 400 (delegated_to user inactive or lacks required role), 403
```

CHAPTER 2.4 — Key Takeaways

› Three parallel approval tracks (Commercial, Legal, Finance) proceed simultaneously — neither blocks the other

› APPR-001: a contract cannot execute without at least one Commercial and one Legal approval

› APPR-002: any track rejection cancels all pending approvals and returns the contract to DRAFT

› APPR-004: SLA breach escalation is automatic — the system escalates, not a human remembering to follow up

› Conditional triggers add approval steps for HIPAA, high-liability, and federal contracts — without conditional triggers, these deals escape required review

CHAPTER 2.5

E-Signature and Contract Execution: Integration Patterns and Legal Evidence

Learning Objectives

• Design the e-signature integration that connects the CLM to electronic signature infrastructure

• Specify the execution event handling that transitions the contract to EXECUTED and triggers downstream processes

• Define the legal evidence requirements for executed contracts including audit trail, tamper-evidence, and certificate of completion

• Configure the multi-party signing workflow for contracts requiring signatures from multiple internal and external signatories

• Establish the failure handling workflow for incomplete or failed signature processes

Process Flow

1. Signature package preparation: when a contract is APPROVED, the primary owner prepares the signature package — selecting signatories (internal and counterparty), setting signing order if required, and attaching supporting documents.

2. Signature request dispatch: the system dispatches signature requests to all signatories through the e-signature integration. Each signatory receives a signing link. The CLM records the dispatch event.

3. Signing in order (if sequential): if signing_order = SEQUENTIAL, the second signer only receives their request after the first signer completes. For PARALLEL signing, all signatories receive requests simultaneously.

4. Signature completion events: as each signatory completes signing, the e-signature provider fires a webhook event. The CLM processes the event, updates the SignatoryStatus, and checks if all required signatories have signed.
5. Full execution: when all required signatures are collected, the e-signature provider issues a completed document with Certificate of Completion. The CLM receives the final document, stores it, and transitions the Contract to EXECUTED.
6. Downstream trigger: execution triggers the extraction pipeline (Chapter 2.1) and fires a contract.executed event on the event bus for downstream system subscribers.

Data Model

```
OBJECT: ContractSignatureRequest
  id                    UUID          REQUIRED
  contract_id           UUID          REQUIRED
  signature_provider    enum          REQUIRED    DOCUSIGN | ADOBE_SIGN |
HELLOSIGN | OTHER
  provider_envelope_id  string        REQUIRED    The provider's
envelope/document ID
  signing_order         enum          REQUIRED    SEQUENTIAL | PARALLEL
  status                enum          REQUIRED    PENDING | IN_PROGRESS |
COMPLETED | VOIDED | DECLINED | EXPIRED
  dispatched_at         ISO 8601 UTC  SYSTEM
  completed_at          ISO 8601 UTC  OPTIONAL
  signed_document_url   string        OPTIONAL    URL to the fully executed PDF
  certificate_url       string        OPTIONAL    URL to the Certificate of
Completion
  expiry_date           ISO 8601 date REQUIRED    Date after which the signing
link expires

OBJECT: ContractSignatory
  id                    UUID          REQUIRED
  signature_request_id  UUID          REQUIRED
  signatory_type        enum          REQUIRED    INTERNAL | COUNTERPARTY
  name                  string        REQUIRED
  email                 string        REQUIRED
  title                 string        OPTIONAL
  signing_sequence      integer       OPTIONAL    For SEQUENTIAL signing; 1 =
first
  status                enum          REQUIRED    PENDING | SENT | VIEWED |
SIGNED | DECLINED
  signed_at             ISO 8601 UTC  OPTIONAL
  ip_address            string        OPTIONAL    IP at time of signing (from
provider)
  authentication_method enum          OPTIONAL    EMAIL | SMS | ACCESS_CODE |
ID_VERIFICATION
```

BUSINESS RULES

ESIG-001: The signed_document_url and certificate_url must be stored in the CLM's document store before the Contract transitions to EXECUTED. The original provider documents are not sufficient — they

must be copied to the CLM's controlled storage.

ESIG-002: For contracts with liability_cap_cents above $1,000,000,000 (i.e., $10M+), the counterparty signatory must use authentication_method = ID_VERIFICATION. Email authentication alone is insufficient.

ESIG-003: A voided or declined signature request cannot be restarted on the same envelope ID. A new ContractSignatureRequest must be created. The original is retained in VOIDED or DECLINED status for audit.

ESIG-004: Signature link expiry_date must be at least 7 days and at most 30 days from dispatch. Shorter expiry prevents adequate review time; longer expiry creates security risk.

ESIG-005: Internal signatories must be authenticated users in the system. External signatories (counterparty) are identified by name and email only.

API Specification

```
POST   /contracts/{id}/signature-requests
       Auth: CONTRACT_EXECUTE
       Request: { signature_provider, signatories: [ContractSignatory],
signing_order, expiry_date }
       Response: 201 — ContractSignatureRequest; triggers dispatch to provider
       Errors: 400, 403, 409 (signature request already active)

GET    /contracts/{id}/signature-requests/{request_id}
       Auth: CONTRACT_READ
       Response: 200 — ContractSignatureRequest with embedded ContractSignatory
statuses

POST   /contracts/{id}/signature-requests/{request_id}/void
       Auth: CONTRACT_EXECUTE
       Request: { reason }
       Response: 200 — ContractSignatureRequest in VOIDED status
       Errors: 400 (already COMPLETED), 403

POST   /webhooks/esignature
       Auth: Webhook secret validation
       Payload: Provider-specific completion event
       Triggers: Updates ContractSignatory status; when all signed → Contract
transitions to EXECUTED → extraction triggered
       Response: 200 (always — prevents provider retry of successfully processed
events)
```

CHAPTER 2.5 — Key Takeaways

› The signed document must be copied to CLM-controlled storage — relying on the provider as primary storage is a compliance risk

› ESIG-001: signed document URL and certificate URL must be stored before EXECUTED transition

— no exceptions
› The e-signature webhook is the execution trigger — missing or failed webhooks must be monitored and re-processed
› Multi-party sequential signing requires monitoring to detect signing process abandonment before all parties have signed
› ESIG-003: voided envelopes must remain in history — they cannot be deleted — the voided signature request is part of the contract audit trail

CHAPTER 2.6

Obligation Tracking: Milestones, Commitments, and Performance Monitoring

Learning Objectives

- Identify the obligation types that must be tracked in an AI product contract and design the obligation data model
- Configure automated obligation monitoring that alerts stakeholders when obligations are approaching or past due
- Specify the obligation performance recording workflow that creates the evidence chain for dispute resolution
- Design the customer-facing obligation status visibility in the customer portal
- Build the obligation analytics that identify patterns of non-performance across the contract portfolio

Process Flow

1. Obligation extraction: when a contract is activated, the extraction pipeline identifies and creates ContractObligation objects for each obligation in the contract. Obligations include: SLA commitments, delivery milestones, minimum consumption commitments, data processing obligations, security review requirements, and renewal notice requirements.
2. Due date calculation: each obligation has a due date calculated from the contract term. Static obligations (e.g., security review within 30 days of execution) are calculated at activation. Recurring obligations (e.g., quarterly business reviews) generate a series of due dates for the contract term.

3. Monitoring and alerts: the monitoring engine checks obligation status daily. Approaching obligations (within the alert window) generate notification events. Overdue obligations generate escalation events.
4. Performance recording: when an obligation is fulfilled, the responsible party records the fulfillment with evidence (meeting notes, reports, audit outputs). The system records the fulfillment date and the evidence reference.
5. Non-performance escalation: overdue obligations that are not fulfilled escalate through the account management hierarchy. If an SLA-type obligation remains unfulfilled, the credit calculation engine is invoked.
6. Customer visibility: the customer portal displays the status of all obligations visible to the customer (SLA performance, milestone status, data processing compliance). Internal obligations are not customer-visible.

Data Model

```
OBJECT: ContractObligation
  id                    UUID          REQUIRED
  contract_id           UUID          REQUIRED
  obligation_type       enum          REQUIRED    SLA_UPTIME | SLA_RESPONSE_TIME
| DELIVERY_MILESTONE | MIN_CONSUMPTION | SECURITY_REVIEW | DATA_PROCESSING |
RENEWAL_NOTICE | QBR | OTHER
  description           string        REQUIRED
  responsible_party     enum          REQUIRED    VENDOR | CUSTOMER | BOTH
  due_date              ISO 8601 date OPTIONAL    null for ongoing obligations
  recurrence            enum          OPTIONAL    NONE | MONTHLY | QUARTERLY |
ANNUAL
  recurrence_end_date   ISO 8601 date OPTIONAL
  alert_days_before     integer       REQUIRED    Days before due date to send
alert; default 14
  status                enum          REQUIRED    PENDING | IN_PROGRESS |
FULFILLED | OVERDUE | WAIVED | DISPUTED
  fulfillment_recorded_at ISO 8601 UTC OPTIONAL
  fulfillment_evidence_url string     OPTIONAL
  customer_visible      boolean       REQUIRED    default false; SLA obligations
default true
  financial_consequence enum          OPTIONAL    SLA_CREDIT |
CONTRACT_TERMINATION_RIGHT | NONE
```

BUSINESS RULES

OBL-001: Obligations with obligation_type = SLA_UPTIME or SLA_RESPONSE_TIME are automatically created from contract SLA fields on activation. They are customer_visible = true by default.

OBL-002: An obligation with financial_consequence = SLA_CREDIT and status = OVERDUE automatically triggers the SLA credit calculation engine (see ROS-09, Chapter 9.5). The credit calculation is initiated without requiring manual intervention.

OBL-003: Obligation WAIVER requires approval from the account owner and the legal owner. A waiver

without dual approval is non-compliant.

OBL-004: Fulfilled obligations cannot be reversed to OVERDUE or PENDING. If a fulfillment was recorded incorrectly, the original fulfillment record is retained and a correction record is created with a reference to the original.

OBL-005: Obligations with customer_visible = true must be reflected in the customer portal within 24 hours of any status change.

API Specification

```
GET    /contracts/{id}/obligations
       Auth: CONTRACT_READ
       Query: status, obligation_type, responsible_party, overdue_only
       Response: 200 — filtered ContractObligation list

POST   /contracts/{id}/obligations/{obligation_id}/record-fulfillment
       Auth: CONTRACT_WRITE
       Request: { fulfillment_evidence_url, fulfillment_notes }
       Response: 200 — updated ContractObligation in FULFILLED status
       Errors: 400 (already FULFILLED), 403, 404

POST   /contracts/{id}/obligations/{obligation_id}/waive
       Auth: Requires CONTRACT_EXECUTE and CONTRACT_LEGAL roles (dual approval)
       Request: { waiver_reason, approved_by_legal_user_id }
       Response: 200 — updated ContractObligation in WAIVED status
       Errors: 400 (missing dual approval), 403

GET    /contracts/obligations/overdue
       Auth: CONTRACT_READ (filtered by account ownership)
       Query: account_id, obligation_type, page, page_size
       Response: 200 — overdue obligations across all accounts visible to
requester
```

CHAPTER 2.6 — Key Takeaways

› Obligations are the contractual commitments that exist beyond the pricing and entitlement terms — SLAs, delivery milestones, security reviews

› OBL-002: SLA_UPTIME and SLA_RESPONSE_TIME obligations with financial_consequence = SLA_CREDIT automatically trigger the credit engine — no human intervention required

› OBL-003: obligation waiver requires dual approval (account owner + legal owner) — single-party waivers are non-compliant

› Recurring obligations generate a series of due dates for the full contract term — these must be created at activation, not on demand

› Customer-visible obligations must appear in the customer portal within 24 hours of any status

change (OBL-005)

CHAPTER 2.7

Amendment Management: Change Control, Proration Impact, and RevRec Modification

Learning Objectives

• Design the amendment lifecycle as a structured change control process attached to the parent contract

• Specify the commercial impact analysis that quantifies the revenue and recognition effects of proposed amendments

• Configure the amendment approval workflow that requires the same governance standards as the original contract

• Define the proration calculation triggered by amendments that change billing terms mid-period

• Specify the RevRec schedule modification required when an amendment creates a contract modification under ASC 606

Process Flow

1. Amendment initiation: a contract amendment is initiated from the parent contract. The Amendment Contract is created with contract_type = AMENDMENT and parent_contract_id referencing the executed parent contract.
2. Change scope definition: the amendment document specifies which terms are being changed. The extraction engine is configured to extract only the changed fields — unchanged terms are inherited from the parent contract.
3. Commercial impact analysis: the system calculates the commercial impact of the proposed changes: the change in ARR, the proration amount for the current billing period, and whether the amendment constitutes a contract modification under ASC 606 (which requires RevRec schedule modification).
4. Amendment approval: the amendment follows the same approval workflow as the original contract, with the approval chain scaled to the amendment's financial impact.
5. Amendment execution: the amendment is signed by both parties. On execution, the amendment's terms supersede the parent contract terms for the specified fields.

6. Downstream update: the billing configuration, entitlements, and RevRec schedule are updated to reflect the amendment terms. Proration is calculated and applied to the next billing run.

Data Model

```
OBJECT: ContractAmendment (subtype of Contract with contract_type = AMENDMENT)
  parent_contract_id    UUID          REQUIRED    FK → Contract.id (must be
EXECUTED)
  amendment_type        enum          REQUIRED    PRICE_CHANGE | SCOPE_CHANGE |
TERM_EXTENSION | EARLY_TERMINATION | SLA_CHANGE | PRODUCT_ADD | PRODUCT_REMOVE
  changed_fields        string[]      REQUIRED    List of Contract fields being
modified
  effective_date        ISO 8601 date REQUIRED    Date from which the amendment
terms apply
  proration_required    boolean       REQUIRED    Calculated by system; true if
amendment changes a billing amount mid-period
  proration_amount_cents integer      OPTIONAL    Credit (negative) or charge
(positive) for current period
  is_ase_606_modification boolean    REQUIRED    Calculated by RevRec engine;
true = full RevRec schedule recalculation required
  arr_delta_cents       integer       OPTIONAL    Change in ARR: positive =
increase, negative = decrease
  tcv_delta_cents       integer       OPTIONAL    Change in TCV
```

BUSINESS RULES
AMEND-001: An amendment's effective_date must be on or after the amendment's execution date. Retroactive amendments (effective before execution) require CFO approval.
AMEND-002: When is_asc_606_modification = true, the RevRec team must review and approve the revised recognition schedule before the amendment's downstream activation is permitted.
AMEND-003: proration_required = true triggers automatic proration credit or charge calculation. The proration amount is displayed to the approver before approval and must be confirmed.
AMEND-004: For PRODUCT_REMOVE amendments, the removed product's entitlements must be deactivated on the amendment's effective_date. Billing for the removed product stops on the effective_date, not the end of the billing period (unless the contract specifies otherwise).
AMEND-005: Multiple amendments to the same parent contract are permitted, but each amendment's changed_fields must be non-overlapping with other pending amendments. Simultaneous amendments to the same field create a conflict that requires serial processing.

API Specification

```
POST   /contracts/{parent_id}/amendments
       Auth: CONTRACT_WRITE
       Request: { amendment_type, changed_fields, effective_date,
```

```
proposed_changes: {} }
       Response: 201 — Amendment Contract; triggers commercial impact analysis
       Errors: 400 (parent not EXECUTED), 403

GET    /contracts/{id}/amendments
       Auth: CONTRACT_READ
       Response: 200 — array of amendment Contracts linked to this parent

GET    /contracts/{amendment_id}/commercial-impact
       Auth: CONTRACT_READ
       Response: 200 — { arr_delta_cents, proration_amount_cents,
proration_required, is_asc_606_modification, affected_entitlements[],
revised_revrec_schedule_preview }
       Errors: 404, 403
```

CHAPTER 2.7 — Key Takeaways

› Amendments are Contracts with contract_type = AMENDMENT — they follow the same lifecycle and governance as the original contract

› is_asc_606_modification = true requires RevRec team review before downstream activation — this is the most important compliance gate in amendment processing

› AMEND-003: proration calculation is shown to the approver before approval — the approver must see the financial impact before deciding

› AMEND-005: overlapping amendments to the same field must be processed serially — simultaneous amendments create billing conflicts

› AMEND-001: retroactive amendments (effective before execution) require CFO approval — retroactive billing changes create prior-period revenue adjustments

CHAPTER 2.8

Renewal Triggers, Auto-Renewal Configuration, and Contract Expiry Management

Learning Objectives

• Design the renewal trigger architecture that fires renewal workflows at the correct time before contract expiry

• Configure auto-renewal logic including notice period management and opt-out handling

• Specify the renewal deal flow that creates a new contract from a renewal deal in the pipeline

• Define the expiry handling workflow for contracts that are not renewed including entitlement deactivation and offboarding

• Build the renewal analytics that support the renewals team's capacity planning and revenue retention tracking

Process Flow

1. Renewal trigger calculation: when a contract is activated, the renewal trigger date is calculated as (end_date minus renewal_alert_days). The renewal_alert_days default is 90 but can be configured per customer segment.
2. Renewal trigger fires: at the trigger date, the system creates a RenewalOpportunity in the CRM, assigns it to the account owner, and creates a RenewalWorkflow tracking object.
3. Auto-renewal notice (if configured): for contracts with auto_renew = true, the system sends the customer a notice at (end_date minus auto_renew_notice_days) informing them of the upcoming auto-renewal and their cancellation window.
4. Renewal deal negotiation: the account owner works the renewal deal. If pricing changes, new contract terms are negotiated. The renewal contract is authored using the renewal template.
5. Renewal contract execution: the renewal contract is approved and executed following the standard contract workflow. The renewal contract references the expiring contract as predecessor.
6. Continuity: when the renewal contract activates, it seamlessly continues the customer's entitlements without a gap. The expiring contract transitions to EXPIRED; the renewal contract's entitlements activate.
7. Expiry without renewal: if no renewal contract is executed before the expiry date, the contract expires and entitlements deactivate. The account transitions to a churned state.

Data Model

```
OBJECT: ContractRenewalConfig
  contract_id           UUID          REQUIRED
  renewal_alert_days    integer       REQUIRED    Days before expiry to trigger
renewal workflow; default 90
  auto_renew            boolean       REQUIRED    Inherits from
Contract.auto_renew
  auto_renew_notice_days integer      CONDITIONAL Required when auto_renew = true
  renewal_term_months   integer       OPTIONAL    default = original contract
term length
  price_escalation_pct  decimal       OPTIONAL    Automatic price increase at
renewal (e.g., 0.05 = 5%); 0 = no escalation
```

```
  renewal_template_id   UUID          OPTIONAL    FK → ContractTemplate.id for
renewal contracts

OBJECT: ContractRenewalWorkflow
  id                    UUID          REQUIRED
  contract_id           UUID          REQUIRED
  status                enum          REQUIRED    TRIGGERED | IN_NEGOTIATION |
RENEWAL_SIGNED | CHURNED | CANCELLED
  triggered_at          ISO 8601 UTC  SYSTEM
  assigned_to           UUID          REQUIRED    FK → User.id (account owner)
  crm_opportunity_id    string        OPTIONAL    CRM opportunity ID created for
this renewal
  renewal_contract_id   UUID          OPTIONAL    FK → Contract.id once renewal
contract exists
  churn_reason          enum          OPTIONAL    PRICE | PRODUCT | COMPETITION |
BUDGET | ACQUISITION | UNKNOWN
  churn_notes           string        OPTIONAL
```

BUSINESS RULES

RENEW-001: A contract's renewal trigger must fire no later than renewal_alert_days before end_date. Missing the trigger is a critical SLA failure — the system must have monitoring on renewal trigger generation to detect and alert on missed triggers.

RENEW-002: Auto-renewal cancellation requests received after the auto_renew_notice_days window has closed cannot be honoured by the system automatically. They must be escalated to the account owner and legal team for manual handling.

RENEW-003: When auto_renew = true and no cancellation is received within the notice period, the system automatically creates a renewal contract from the renewal template and routes it for approval with the auto-renewal terms. Human review is still required before execution.

RENEW-004: A contract that reaches end_date without either a RENEWAL_SIGNED workflow status or a confirmed extension transitions to EXPIRED at midnight UTC on end_date. Entitlement deactivation fires within 1 hour of the EXPIRED transition.

RENEW-005: price_escalation_pct creates a pricing change — the renewal contract's pricing plan must reflect the escalated price, and a PricingChangeRequest must be created for the price change if the escalated price exceeds the standard authority level.

API Specification

```
GET    /contracts/{id}/renewal-config
       Auth: CONTRACT_READ
       Response: 200 — ContractRenewalConfig

PATCH  /contracts/{id}/renewal-config
       Auth: CONTRACT_WRITE
       Request: Any mutable field
       Response: 200 — updated config
       Errors: 400, 403
```

```
GET    /contracts/{id}/renewal-workflow
       Auth: CONTRACT_READ
       Response: 200 — ContractRenewalWorkflow

PATCH  /contracts/{id}/renewal-workflow
       Auth: CONTRACT_WRITE
       Request: { status, renewal_contract_id, churn_reason, churn_notes }
       Response: 200 — updated workflow
       Errors: 400, 403

GET    /renewals/upcoming
       Auth: CONTRACT_READ (filtered by account ownership)
       Query: days_until_expiry, auto_renew, status, page, page_size
       Response: 200 — upcoming renewals sorted by urgency
```

CHAPTER 2.8 — Key Takeaways

› The renewal trigger must fire no later than renewal_alert_days before end_date — monitoring on trigger generation is required (RENEW-001)

› Auto-renewal creates a renewal contract that still requires human approval before execution — auto-renewal is not automatic execution

› RENEW-004: contracts that reach end_date without renewal transition to EXPIRED at midnight UTC — entitlement deactivation fires within 1 hour

› price_escalation_pct creates a pricing change that must go through the PricingChangeRequest workflow if above authority thresholds

› Churn reasons must be recorded for every CHURNED renewal workflow — this data feeds the retention analytics and renewal risk model

CHAPTER 2.9

Data Integrity: Contract Version History, Audit Trail, and Tamper-Evidence

Learning Objectives

- Design the immutable audit trail architecture that records every change to every contract object
- Specify the tamper-evidence requirements for executed contracts and their supporting documents

• Configure the version history display that allows authorized users to see the complete change history for any contract

• Define the data retention requirements for contract records including post-expiry retention periods

• Build the contract data export capability that produces legally admissible records for dispute resolution

Process Flow

1. Every mutation to a Contract or related object (ContractExtractionField, Redline, ContractApproval, ContractObligation) creates an immutable audit log entry. The entry records: the object type and ID, the field name, the old value, the new value, the user who made the change, and the timestamp.
2. Executed contract documents are stored with cryptographic hashing. The SHA-256 hash of the signed PDF is recorded at the time of storage. Any subsequent modification to the stored document would produce a different hash — this is the tamper-evidence mechanism.
3. Version history is accessible through the API and the contract detail view. All previous versions of the contract document are retained and accessible. Document deletion is prohibited.
4. The audit trail is append-only — records cannot be modified or deleted. The audit trail database is separate from the operational database and has restricted write access (only the audit logging service can write to it).
5. Data retention: executed contracts are retained for a minimum of 7 years after contract expiry. TERMINATED contracts are retained for the longer of 7 years or the applicable statute of limitations.

Data Model

```
OBJECT: ContractAuditEntry
  id                    UUID          REQUIRED    Immutable once created
  contract_id           UUID          REQUIRED
  object_type           string        REQUIRED    e.g., "Contract",
"ContractExtractionField", "Redline"
  object_id             UUID          REQUIRED
  action                enum          REQUIRED    CREATED | UPDATED |
STATUS_CHANGED | DELETED (soft delete only)
  field_name            string        OPTIONAL    Null for CREATED/DELETED
actions; populated for UPDATED
  old_value             string        OPTIONAL    JSON-serialized old value; null
for CREATED
  new_value             string        OPTIONAL    JSON-serialized new value; null
for DELETED
  actor_type            enum          REQUIRED    USER | SYSTEM | AGENT
  actor_id              UUID          REQUIRED    User ID, system service ID, or
agent ID
  timestamp             ISO 8601 UTC  REQUIRED    Set by audit logging service;
cannot be overridden
  ip_address            string        OPTIONAL    For USER actions
```

```
  session_id            string        OPTIONAL

OBJECT: ContractDocumentHash
  contract_id           UUID          REQUIRED
  contract_version_id   UUID          REQUIRED
  document_type         enum          REQUIRED    DRAFT | EXECUTED | CERTIFICATE
| AMENDMENT
  sha256_hash           string        REQUIRED    256-bit hash of the document at
time of storage
  stored_at             ISO 8601 UTC  SYSTEM
  verified_at           ISO 8601 UTC  OPTIONAL    Last time hash was verified
against stored document
  verification_status   enum          OPTIONAL    INTACT | TAMPERED | UNVERIFIED
```

BUSINESS RULES

INTG-001: ContractAuditEntry records are immutable. No API endpoint may modify or delete an audit entry. The audit database is protected by a separate access control layer from the operational database.

INTG-002: ContractDocumentHash must be generated at the time of document storage — not retroactively. A document stored without a hash cannot be considered tamper-evident.

INTG-003: Document hash verification must run on a scheduled basis for all EXECUTED contract documents. Failed verification (hash mismatch) must trigger an immediate security incident.

INTG-004: Contract records must be retained for 7 years minimum after contract expiry. The data retention policy must be enforced by automated archival — not manual deletion requests.

INTG-005: Export of contract records for legal proceedings must include the complete audit trail, all document versions with their hashes, and the Certificate of Completion from the e-signature provider.

API Specification

```
GET    /contracts/{id}/audit-trail
       Auth: CONTRACT_AUDIT_READ (restricted role)
       Query: object_type, actor_id, from_timestamp, to_timestamp, page,
page_size
       Response: 200 — paginated ContractAuditEntry list in chronological order

GET    /contracts/{id}/document-hashes
       Auth: CONTRACT_AUDIT_READ
       Response: 200 — array of ContractDocumentHash for all versions

POST   /contracts/{id}/document-hashes/{hash_id}/verify
       Auth: CONTRACT_AUDIT_READ
       Triggers: Re-computes hash of stored document and compares to stored hash
       Response: 200 — { verification_status, computed_hash, stored_hash,
verified_at }
       Errors: 404, 403

GET    /contracts/{id}/export
```

```
       Auth: CONTRACT_AUDIT_READ
       Format: ZIP archive containing: executed document, all versions, audit
trail CSV, document hashes, certificate of completion
       Response: 202 Accepted — { export_job_id }; download link provided on
completion
       Errors: 403, 404
```

CHAPTER 2.9 — Key Takeaways
› ContractAuditEntry records are immutable and stored in a separate database with restricted write access
› SHA-256 document hashing (INTG-002) is the tamper-evidence mechanism — hash must be generated at storage time
› INTG-003: scheduled hash verification is mandatory — a document with a failing hash verification is a security incident
› INTG-004: 7-year minimum retention after expiry is enforced by automated archival — manual deletion is prohibited
› The complete audit trail + all document versions + hashes + Certificate of Completion is the export package for legal proceedings

CHAPTER 2.10

AI Agents for Contract Lifecycle Management

Four AI agents that accelerate CLM operations — from drafting through obligation monitoring to renewal intelligence

Agent 1: Contract Drafting Agent

Specification	
Attribute	**Specification**
Function	Accelerates contract authoring by populating templates from deal records, suggesting appropriate clauses based on deal attributes, and flagging missing required fields before the contract is sent for review
Data access	Read access to ContractTemplates, ClauseLibrary, Deal records from CRM, Account data, PricingPlans

Tools	template_selector, variable_populator, clause_recommender, consistency_checker, missing_field_detector
Guardrails	May not modify non_negotiable_clause_ids or their text · Clause recommendations are suggestions — the deal desk must select or reject each one · Cannot submit a contract for approval — that is a human action · Cannot accept or reject counterparty redlines
Outputs	Pre-populated contract draft with confidence indicators on each auto-populated variable. Clause recommendation list with rationale. Internal consistency check report. Missing field checklist.
Value	Reduces average contract drafting time from 4–6 hours to 30–45 minutes for standard deals. Eliminates common errors from manual template variable population.

Agent 2: Contract Intelligence and Redline Analysis Agent

Specification	
Attribute	**Specification**
Function	Analyses counterparty redlines to classify risk, identify commercial term impacts, and suggest responses based on the clause library's approved deviations
Data access	Read access to Contracts, Redlines, ClauseLibrary (including approved_deviations), historical negotiation outcomes, ContractApprovalMatrix
Tools	redline_classifier, commercial_impact_analyser, approved_deviation_matcher, risk_scorer, response_suggester
Guardrails	READ-ONLY on all contract objects — cannot accept, reject, or modify redlines · Suggestions are for human review — not auto-accepted · Cannot assess legal risk for novel clause structures — escalates to legal for anything not in the clause library · Cannot generate new contract language — only suggests from approved_deviations
Outputs	Redline triage report: each redline classified by risk, commercial impact, and whether it matches an approved deviation. Suggested response for each redline (accept / reject / counter with approved deviation text). Summary of commercial term changes if all redlines are accepted.
Value	Reduces legal review time for standard redlines by 60–70%. Ensures approved deviations are consistently applied across all negotiations. Surfaces commercial term impacts that deal desk may miss in dense legal documents.

Agent 3: Obligation Monitoring Agent

Specification	
Attribute	**Specification**
Function	Continuously monitors ContractObligation status across all active contracts and fires alerts when obligations are approaching due dates or overdue
Data access	Read access to ContractObligation, Contract, Account — across all active contracts in the portfolio
Tools	obligation_status_scanner, due_date_calculator, alert_dispatcher, sla_performance_tracker, credit_trigger_evaluator
Guardrails	READ-ONLY — cannot record obligation fulfillment or waiver · SLA credit triggers require human confirmation before credit is issued · Cannot modify obligation due dates or recurrence schedules
Outputs	Real-time obligation health dashboard. Daily alert digest for account owners. Weekly portfolio summary for CS and RevOps leadership. SLA performance report for billing team (inputs to credit calculation).
Value	Prevents obligation SLA breaches from going unnoticed. Ensures SLA credits are calculated and issued promptly when breaches occur. Provides early warning for at-risk accounts.

Agent 4: Renewal Intelligence Agent

Specification	
Attribute	**Specification**
Function	Analyses the renewal portfolio to identify at-risk accounts, recommend renewal pricing, and prepare renewal deal packages for the account team
Data access	Read access to Contracts, ContractRenewalWorkflow, Account health data, usage data, billing history, discount history
Tools	renewal_risk_scorer, usage_trend_analyser, pricing_recommender, renewal_package_generator, churn_predictor
Guardrails	READ-ONLY on contract and billing data · Pricing recommendations are suggestions — cannot modify PricingPlans or create PricingChangeRequests · Cannot initiate renewal contracts — that is a human action · Churn risk scores are model outputs — human account owner interprets and acts
Outputs	Renewal risk score for each account (30–60–90 day buckets). Renewal package draft for each upcoming renewal: recommended pricing, comparison to current terms, usage-based value story. Churn risk report for CS leadership.
Value	Enables account team to prioritize renewal engagement on highest-risk accounts. Automates renewal package preparation, reducing time-to-first-renewal-conversation from weeks to days.

Chapter 2.10 — Key Takeaways
› Four agents cover drafting acceleration, redline analysis, obligation monitoring, and renewal intelligence
› The Drafting Agent reduces standard contract authoring from 4–6 hours to 30–45 minutes — the highest-ROI agent in this module
› The Redline Analysis Agent only suggests approved deviations — it cannot generate novel contract language
› The Obligation Monitoring Agent fires SLA credit evaluation automatically on breach — human confirms before credit is issued
› The Renewal Intelligence Agent produces churn risk scores and renewal packages — human account owner acts on these outputs

APPENDIX A

AI Agent Reference

Consolidated specification for all four AI agents in ROS-02

This appendix provides the consolidated reference for all AI agents specified in ROS-02. All agents are READ-ONLY on contract objects unless otherwise specified. Agent actions are logged to the audit trail.

> GOVERNANCE PRINCIPLE
>
> **CLM agents accelerate human work — they do not replace human judgment on legal and commercial decisions**
>
> Contracts are legally binding documents. No AI agent in this module may execute contracts, accept redlines, approve amendments, or waive obligations. All agents produce analysis, suggestions, and alerts that human owners act upon. This boundary is firm and non-negotiable.

CLOSING

The Contract Governs Everything

This module specifies how to govern the contract.

The contract lifecycle, as specified in this book, is the commercial intelligence backbone of the revenue operations platform. Every object defined here — the Contract, the ContractExtractionField, the ContractObligation, the ContractAmendment — is designed to be machine-readable, not just human-readable.

The distinction matters in practice. A contract that lives as a PDF answers the question "what did we agree?" when a human searches for it and reads it. A contract whose terms are captured in the canonical objects defined here answers the same question automatically, continuously, and at scale — for every customer, at every billing run, without human intervention.

The CLM architecture specified in this book creates three types of commercial value. It prevents billing errors by ensuring that the billing configuration is derived from the contract, not maintained in parallel with it. It enables revenue assurance by creating the audit trail that proves the billing configuration matches the contract when customers dispute. And it enables commercial intelligence by making contract data available to the analytics layer, the forecasting models, and the renewal risk engine.

The effort required to build a CLM to this specification is substantial. The effort required to resolve billing disputes, credit customers for misconfiguration errors, and defend revenue recognition positions without this infrastructure is typically larger. Build the foundation.

In Module 3 (ROS-03), the customer onboarding module picks up after the contract is executed — capturing the customer's operational data and provisioning their entitlements. In Module 4 (ROS-04), the deal desk uses the CLM as its governing authority for commercial terms. In Module 5 (ROS-05), order management is driven entirely by the executed contract.

The contract governs everything. This module specifies how to govern the contract.

> ***"If it is not in the contract data model, the billing system will not honour it."***
