



CREDITX COMPLETE [Consumer] AGENT CATALOG

Master List - All [agent_code_foundation] Agents with Full Specifications; 1-10

ENGINE 1: OUTCOME ENGINE (6 Agents)

Agent 1: Plan Generation Agent

text

```
agent_id: outcome.plan_generation.v1 name: Plan Generation
Agent type: assistant risk_level: medium faces: [consumer]
milestone: M3 purpose: | Generate personalized financial
improvement plans based on consumer goals, current
snapshot, and predictive modeling input_entities: -
ConsumerSnapshot (credit_score, debt_summary,
payment_history) - ConsumerGoals (target_score, timeline,
constraints) - ConsumerConstraints (budget, risk_tolerance)
output_entities: - SavingsPlan (draft, not committed) -
ActionItems (prioritized list) - PredictionOutputs
(expected trajectory) tools_required: -
get_consumer_snapshot - get_consumer_goals -
compute_prediction - draft_plan business_logic: 1. Fetch
consumer credit snapshot 2. Retrieve active consumer goals
3. Run prediction model for various scenarios 4. Generate
optimal action plan 5. Draft plan (does NOT auto-commit)
error_handling: - Missing snapshot → Request consumer re-
sync - No goals set → Suggest default goals - Prediction
failure → Use fallback heuristics requires_human_review:
false max_auto_actions: N/A (read-only)
```

Agent 2: Outcome Evaluation Agent

text

```
agent_id: outcome.evaluation.v1 name: Outcome Evaluation
Agent type: operator risk_level: low faces: [internal]
milestone: M4 purpose: | Continuously evaluate consumer
outcomes against predictions, update tracking metrics, and
flag variance anomalies input_entities: -
ConsumerOutcomeSummary (actual vs predicted) -
CampaignOutcomeSummary (aggregated results) - TimePeriod
(evaluation window) output_entities: - OutcomeTracks
(updated records) - PredictionOutputs (calibration scores)
- CalibrationMetrics (model accuracy) tools_required: -
get_outcome_summary - update_outcome_tracks -
compute_calibration_metrics - create_audit_event
business_logic: 1. Get all outcomes in time window 2.
Compare actual vs predicted improvements 3. Calculate
variance and confidence intervals 4. Update outcome_tracks
table 5. Flag significant deviations (>20% variance) 6.
Generate calibration report error_handling: - Incomplete
data → Log warning, skip consumer - Negative variance →
Flag for investigation - Missing predictions → Backfill if
possible requires_human_review: false max_auto_actions:
unlimited (internal only)
```

Agent 3: Campaign Tuning Agent

text

```
agent_id: outcome.campaign_tuning.v1 name: Campaign Tuning
Agent type: operator risk_level: medium faces: [internal]
milestone: M3 purpose: | Analyze campaign performance and
recommend configuration adjustments to improve outcomes
input_entities: - CampaignOutcomeSummary (performance
metrics) - CampaignConfig (current settings) -
SegmentAnalysis (performance by segment) output_entities:
- CampaignConfigRecommendations (draft changes) -
ExpectedImpact (projected improvement) - RiskAssessment
(change impact) tools_required: - get_campaign_summary -
analyze_segment_performance - draft_config_changes -
simulate_config_impact business_logic: 1. Fetch campaign
performance over 30/60/90 days 2. Segment performance by
consumer attributes 3. Identify underperforming segments
```

```
4. Generate tuning recommendations (e.g., adjust targeting)
5. Simulate impact of changes    6. Draft config change
request error_handling:    - Insufficient data → Wait for
more results    - Conflicting signals → Flag for human
analysis    - High-risk changes → Auto-require review
requires_human_review: true (changes affect real campaigns)
max_auto_actions: 0 (draft only)
```

Agent 4: Consumer Progress Tracker Agent

```
text
agent_id: outcome.progress_tracker.v1 name: Consumer Progress
Tracker Agent type: assistant risk_level: low faces:
[consumer, internal] milestone: M3 purpose: | Track
individual consumer progress toward goals and generate
milestone notifications input_entities:    - ConsumerGoals
(active goals)    - ConsumerSnapshots (historical)    -
OutcomeTracks (progress data) output_entities:    -
ProgressSummary (current status)    - MilestoneEvents
(achievements)    - MotivationalInsights (personalized
messages) tools_required:    - get_consumer_snapshots    -
get_consumer_goals    - calculate_progress    -
enqueue_notification business_logic:    1. Fetch consumer's
active goals    2. Get historical snapshots (last 6 months)
3. Calculate progress % toward each goal    4. Detect
milestones (25%, 50%, 75%, 100%)    5. Generate personalized
progress summary    6. Queue celebration/encouragement
notifications error_handling:    - No recent snapshots →
Prompt consumer to sync    - Goals expired → Archive and
suggest new ones    - Negative progress → Flag for support
outreach requires_human_review: false max_auto_actions:
unlimited (notifications only)
```

Agent 5: Prediction Calibration Agent

```
text
agent_id: outcome.calibration.v1 name: Prediction Calibration
Agent type: operator risk_level: low faces: [internal]
```

```

milestone: M4 purpose: | Analyze prediction accuracy and
recommnd model retraining or recalibration
input_entities: - PredictionOutputs (all historical
predictions) - OutcomeTracks (actual results) -
ModelVersion (current model metadata) output_entities: -
CalibrationReport (accuracy metrics) -
RecalibrationRecommendations (model adjustments) -
SegmentDriftAnalysis (performance by segment)
tools_required: - get_prediction_outputs -
get_outcome_tracks - compute_calibration_metrics -
detect_model_drift business_logic: 1. Fetch all
predictions from last 90 days 2. Match predictions with
actual outcomes 3. Calculate MAE, RMSE, confidence
intervals 4. Segment analysis (by demographics, score
bands) 5. Detect model drift (statistical tests) 6.
Recommend retraining if accuracy < 85% error_handling: -
Insufficient matches → Wait for more data - Catastrophic
drift → Immediate alert - Segment bias detected → Flag for
fairness review requires_human_review: true (model changes
are high-stakes) max_auto_actions: 0 (recommendations only)

```

Agent 6: Scenario Comparison Agent

```

text
agent_id: outcome.scenario_comparison.v1 name: Scenario
Comparison Agent type: assistant risk_level: low faces:
[consumer] milestone: M3 purpose: | Generate and compare
multiple "what-if" scenarios for consumers to help them
make informed decisions input_entities: - ConsumerSnapshot
(current state) - ScenarioDefinitions (user-provided
scenarios) - ConsumerGoals (target outcomes)
output_entities: - ScenarioResults (predicted outcomes per
scenario) - ComparisonTable (side-by-side analysis) -
Recommendations (optimal path) tools_required: -
get_consumer_snapshot - compute_prediction (multiple calls)
- compare_scenarios - generate_recommendation
business_logic: 1. Receive 2-5 scenarios from consumer
(e.g., "pay off card A" vs "pay off card B") 2. Run
prediction for each scenario 3. Compare outcomes (score
impact, time to goal, cost) 4. Rank scenarios by
effectiveness 5. Generate human-readable comparison 6.

```

```
Recommend top scenario with reasoning error_handling: -  
Invalid scenario → Return validation error - Prediction  
fails → Use fallback estimates - Tied scenarios → Present  
both options requires_human_review: false max_auto_actions:  
N/A (read-only)
```

ENGINE 2: RIGHTS & TRUST ENGINE (6 Agents)

Agent 7: Consent Scope Assistant

```
text  
agent_id: rights.consent_scope.v1 name: Consent Scope  
Assistant type: assistant risk_level: high faces: [consumer,  
partner] milestone: M4 purpose: | Help consumers and  
partners understand consent scopes and suggest appropriate  
permissions for use cases input_entities: -  
DataRightsSummary (existing consents) - PartnerProfile  
(requesting partner info) - Purpose (intended use case)  
output_entities: - SuggestedScopes (recommended  
permissions) - Explanations (plain-language descriptions)  
- RiskAssessment (privacy impact) tools_required: -  
get_rights_summary - explain_scope - assess_privacy_risk  
- generate_consent_language business_logic: 1. Analyze  
partner's stated purpose 2. Map purpose to minimum required  
data types 3. Check if existing consents cover the need  
4. Suggest new scopes if needed (least-privilege) 5.  
Generate plain-language explanations 6. Assess privacy risk  
(low/medium/high) 7. Present options to consumer  
error_handling: - Vague purpose → Request clarification -  
High-risk scope → Require explicit opt-in - Over-scoped  
request → Suggest alternatives requires_human_review: true  
(consent is sensitive) max_auto_actions: 0 (advisory only)
```

Agent 8: Rights Request Orchestrator

```
text  
agent_id: rights.request_orchestrator.v1 name: Rights Request
```

```
Orchestrator type: ambassador risk_level: high faces:
[consumer, internal] milestone: M5 purpose: | Execute
consumer data rights requests (export, delete, non-use)
across all systems input_entities: - RightsRequest
(consumer request details) - DataRightsSummary (current
state) - AffectedSystems (systems holding data)
output_entities: - ConsentEvent (revocation/update) -
AuditEvent (compliance record) - StatusUpdate (progress
notification) tools_required: - execute_rights_action -
create_audit_event - enqueue_notification -
verify_deletion_cascade business_logic: 1. Validate
consumer identity 2. Parse rights request (export/delete/
restrict) 3. Identify all systems holding consumer data
4. Execute actions across systems (with retries) 5. Verify
completion 6. Create audit trail 7. Notify consumer of
completion error_handling: - Partial failure → Rollback
and retry - External system timeout → Queue for manual
handling - Irreversible action → Require explicit
confirmation requires_human_review: true (deletion is
irreversible) max_auto_actions: 1 per 24 hours
```

Agent 9: Dispute Advocacy Agent

text

```
agent_id: rights.dispute_advocacy.v1 name: Dispute Advocacy
Agent type: ambassador risk_level: high faces: [consumer,
internal] milestone: M4 purpose: | Draft and submit credit
bureau dispute letters on behalf of consumers
input_entities: - DisputeSummary (items to dispute) -
AdvocacyMandate (consumer authorization) - EvidenceDocs
(supporting documentation) output_entities: -
DisputeLetter (generated letter) - PortalSubmission (bureau
tracking info) - AuditEvent (compliance record)
tools_required: - draft_letter - submit_dispute -
create_audit_event - track_dispute_status business_logic:
1. Verify consumer has granted advocacy rights 2. Analyze
dispute items (errors, inaccuracies) 3. Select appropriate
letter template 4. Personalize letter with consumer details
5. Attach evidence documents 6. Submit to credit bureau via
API/portal 7. Track submission and save reference number
8. Schedule follow-up (30 days) error_handling: - Missing
```

evidence → Request from consumer - Bureau API down → Fall back to postal mail - Submission rejected → Escalate to human advocate requires_human_review: true (external communications) max_auto_actions: 3 per month per consumer

Agent 10: Fairness Analysis Agent

text

```
agent_id: rights.fairness_analysis.v1 name: Fairness Analysis
Agent type: operator risk_level: medium faces: [internal]
milestone: M3 purpose: | Analyze fairness metrics across
demographic segments and flag potential disparate impact
input_entities: - FairnessSummary (metrics by segment) -
UnderwritingSummary (decision data) -
PredictionOutputsSummary (model outputs) output_entities:
- FairnessReport (statistical analysis) -
InvestigationRecommendations (flagged issues) -
MitigationSuggestions (corrective actions) tools_required:
- get_fairness_summary - compute_fairness_metrics -
detect_disparate_impact - recommend_mitigation
business_logic: 1. Fetch outcomes by protected segments
(race, gender, age) 2. Compute disparate impact ratios 3.
Run statistical significance tests 4. Flag segments with
ratio < 0.8 (4/5ths rule) 5. Analyze root causes (data,
model, process) 6. Recommend mitigations (reweighting,
feature removal) 7. Generate compliance report
error_handling: - Insufficient data → Flag for future
analysis - Significant disparate impact → Immediate
escalation - Data quality issues → Request data audit
requires_human_review: false (reporting only)
max_auto_actions: unlimited (internal analysis)
```

****Template For Agent Production: //**

Tailor_all_template_foundations_for_agent_production_live_function_and_purpose_with_enterprise_grade_builds**

create agent [___] agent for a production live build using the provided template. Tailor the template to agent [___]'s design function and purpose for a production live build of enterprise grade quality designed for the creditx platform: # AGENT [X]: [AGENT NAME]

Document Control

****Version:** [X.0.0]-production**

****Status:** Production Ready | In Development | Deprecated**

****Owner:**** [Team Name]
****Last Updated:**** YYYY-MM-DD
****Review Required By:**** Engineering Lead, Security Architect, SRE Lead

1. AGENT REGISTRY ENTRY

```
```sql
INSERT INTO agent_registry (
 agent_id,
 name,
 version,
 engine,
 faces,
 agent_type,
 scope,
 input_entities,
 output_entities,
 required_tools,
 risk_level,
 requires_human_review,
 max_auto_actions,
 status,
 milestone,
 config_json,
 owner_team
) VALUES (
 '[engine].[name].v[X]',
 '[Human-readable Name]',
 '[X.0.0]',
 '[engine_name]',
 ARRAY['[face1]', '[face2]'],
 '[assistant|operator|ambassador]',
 '[Brief description of agent responsibility]',
 ARRAY['[InputEntity1]', '[InputEntity2]'],
 ARRAY['[OutputEntity1]', '[OutputEntity2]'],
 ARRAY['[tool1]', '[tool2]'],
 '[low|medium|high]',
 [true|false],
 [NULL|number],
 'active',
 '[M0-M5]',
 '{[config_json]}::jsonb',
 '[team-name]'
);
```