🔄 Prime Broker Trade Assignment Engine

This document outlines a full solution design for assigning a batch of fixed-income trades to Prime Brokerage (PB) accounts under hard constraints and optimization priorities. The process ensures constraint compliance while attempting to minimize business-critical metrics like financing cost.

🚩 Problem Context

In managing multiple PB accounts, each trade must be assigned to an account that meets all risk and compliance constraints. Accounts also differ in how they finance trades, which affects cost and exposure. Our objective is to find an assignment that:

- Satisfies **all hard constraints** (e.g., positive equity, leverage ratio caps)
- Optimizes account metrics (e.g., total financing cost) according to input priorities

Constraints vary by account and metric; optimization priorities are also account-specific. This makes the problem an instance of **constrained multi-objective trade assignment**.

📥 Inputs

- **Trades** (0 to 200 per batch)
  - Side (buy/sell)
  - Security Name and Type (corporate or government bond)
  - Quantity, Price, Accrued Interest
- **Accounts** (2–5)
  - Initial positions and current metrics
  - Per-account metric definitions (1–20 metrics)
  - Metric coefficients `ω[a][m][s]` (weight by security/account/metric)
  - Hard constraints `C[a][m]` (optional)
  - Optimization priorities `OP[a][m]` (optional)
- **Configuration**
  - Objective weight map: user-defined prioritization of metrics
  - Solver timeout (e.g., 10 seconds)

📤 Outputs

- Account assignment for each trade
- Final metric values for each account
- Solver status (optimal, feasible, or failed)

🧠 Algorithm Design

🎯 Goals

1. **Primary:** Ensure all metric-based constraints are satisfied

2. **Secondary:** Optimize metric scores according to investment team priorities

💡 Rationale for Optimization Model

The assignment problem is best formulated as a **Mixed Integer Program (MIP)**, solved using Google OR-Tools. MIPs allow for binary decision variables (assign trade to account), hard linear constraints (per-metric), and objective functions over weighted sums.

📉 Metric Computation

```
M[a][m] = ∑ x[i][a] × V[i] × ω[a][m][s(i)]
```

- `x[i][a] ∈ {0,1}`: trade `i` assigned to account `a`
- `V[i]`: trade value (price × quantity ± accrued interest)
- `ω[a][m][s]`: metric sensitivity per account/metric/security

✅ Constraints

```
M[a][m] ≤ C[a][m]  (if upper bound exists)
M[a][m] ≥ C[a][m]  (if lower bound exists)
```

🏆 Objective

```
maximize ∑ priority[a][m] × M[a][m]
```

Only *unconstrained* metrics with non-zero priorities are used in the objective.

🔁 Fallback Strategy

1. Try full optimization (all constraints and objectives)
2. If infeasible or timeout, remove objectives → solve only for feasibility

---

🧱 Code & Architecture

- `build_model(trades, accounts, constraints, priorities)`: returns OR-Tools solver and variable objects
- `solve_model(solver)`: attempts full solve with timeout
- `fallback_feasibility(solver)`: strips objective and retries
- `extract_solution(x)`: maps binary variables to output assignments

📦 Modular Components

- 📊 **Trade Loader:** parses trade files and builds position values
- 📐 **Metric Engine:** computes all per-account metric adjustments
- ⚙️ **Constraint Manager:** generates constraints and prioritizes objectives
- 🧮 **OR-Tools Solver:** runs optimization and fallback
- 📤 **Reporter:** outputs assignment, metrics, and diagnostics

---

🚦 Boundary Conditions

- 0 trades: return current metrics, no changes
- Conflicting constraints: solver will return infeasible (log and alert)
- Multiple metrics apply to same trade in same account (aggregated correctly)

💼 Business Practices & Longevity

🔍 Monitoring

- Audit logs of solver inputs/outputs
- Feasibility fallback logs and alerts
- Daily constraint breach tracking

🧪 Testing

- Unit tests for each component
- Scenario-based validation (e.g., all buys, all sells, constraint edge cases)

📈 Operationalization

- Run as a daily batch or triggered pipeline (e.g., Airflow or GitHub Actions)
- Dry-run mode for traders to preview impact
- Configurable via YAML or JSON files for constraints, weights, and limits

🔄 Maintenance

- Modular code to support adding new metrics or security types
- Periodic review of metric weightings with investment team

🧠 Pseudocode Summary

```
for each trade i:
    for each account a:
        calculate V[i] based on trade data
        for each metric m in a:
            delta_M[a][m] += x[i][a] * V[i] * ω[a][m][security]

create binary vars x[i][a] ∈ {0, 1}
add constraint: ∑ x[i][a] over a = 1 (each trade assigned once)
```

```
for all constrained metrics M[a][m]:
    add constraint M[a][m] ≤ or ≥ bound

objective = ∑ priority[a][m] ∗ M[a][m]  (only unconstrained metrics)

try solver.Solve()
if failed or timeout:
    remove objective
    try solver.Solve()
```

📌 Conclusion

This design ensures compliance with Prime Broker constraints while minimizing business costs. It is modular, auditable, and robust to scale and input volatility. Through clear separation of trade logic, metric management, and optimization, it supports future adaptability and reliability in live operations.