

Tax Ready Bookkeeping

Dual-Language Policy: Sample Document

Tax Ready Bookkeeping | projectbits.com/taxready

What Is Dual-Language Policy?

A dual-language policy expresses the same business rule in two forms: 1. **Human Language** - English that people understand 2. **Machine Language** - Code that systems can execute

This ensures policies are both understood by your team AND enforced by your systems.

Policy 1: Invoice Approval

Human Language (for the Policy Manual)

Invoice Approval Policy

All invoices must be approved before payment according to the following hierarchy:

1. Invoices up to \$500 may be approved by any authorized employee
2. Invoices between \$500 and \$2,500 require manager approval
3. Invoices between \$2,500 and \$10,000 require department head approval
4. Invoices over \$10,000 require owner/executive approval and a purchase order on file
5. All invoices over \$1,000 must have supporting documentation attached

Exceptions require written justification and next-level approval.

Machine Language (for the Policy Engine)

```
# Invoice Approval Policy - OPA Rego format
```

```
package invoice.approval
```

```
# Define approval thresholds
approval_level := "employee" if {
    input.amount <= 500
}
```

```

approval_level := "manager" if {
    input.amount > 500
    input.amount <= 2500
}

approval_level := "department_head" if {
    input.amount > 2500
    input.amount <= 10000
}

approval_level := "executive" if {
    input.amount > 10000
}

# Approval requirements
requires_po := true if {
    input.amount > 10000
}

requires_documentation := true if {
    input.amount > 1000
}

# Validation checks
findings contains msg if {
    input.amount > 1000
    not input.has_documentation
    msg := "Invoice over $1,000 requires supporting documentation"
}

findings contains msg if {
    input.amount > 10000
    not input.has_purchase_order
    msg := "Invoice over $10,000 requires purchase order"
}

findings contains msg if {
    approval_level == "manager"
    input.approver_level == "employee"
    msg := "Invoice requires manager approval (amount > $500)"
}

findings contains msg if {
    approval_level == "department_head"
    input.approver_level != "department_head"
    input.approver_level != "executive"
    msg := "Invoice requires department head approval (amount > $2,500)"
}

```

```
}

findings contains msg if {
    approval_level == "executive"
    input.approver_level != "executive"
    msg := "Invoice requires executive approval (amount > $10,000)"
}
```

Policy 2: Vendor Onboarding

Human Language (for the Policy Manual)

Vendor Onboarding Policy

No payment shall be made to any vendor until proper onboarding is complete:

1. A completed W-9 must be on file before first payment
 2. Vendor contact information (phone, email, address) must be verified
 3. Vendors receiving \$600+ annually must have 1099 tracking enabled
 4. Payment method changes require verification via a known phone number
 5. New vendors exceeding \$5,000 in anticipated annual spend require owner approval
-

Machine Language (for the Policy Engine)

Vendor Onboarding Policy - OPA Rego format

```
package vendor.onboarding

# Check W-9 requirement
w9_required := true if {
    input.vendor.is_new
}

w9_required := true if {
    not input.vendor.w9_on_file
}

# Check contact verification
contact_verified := true if {
    input.vendor.phone_verified
    input.vendor.email_verified
    input.vendor.address_complete
}

# Findings for policy violations
findings contains msg if {
    input.action == "payment"
```

```

    not input.vendor.w9_on_file
    msg := sprintf("Cannot process payment: W-9 missing for vendor '%s'",
                  [input.vendor.name])
}

findings contains msg if {
    input.action == "payment"
    not contact_verified
    msg := sprintf("Vendor '%s' contact information incomplete or unverified",
                  [input.vendor.name])
}

findings contains msg if {
    input.vendor.annual_payments_expected > 5000
    input.vendor.is_new
    not input.vendor.owner_approved
    msg := sprintf("New vendor '%s' with expected spend >$5,000 requires owner approval",
                  [input.vendor.name])
}

findings contains msg if {
    input.vendor.payment_method_changed
    not input.vendor.payment_change_verified
    msg := sprintf("Payment method change for '%s' requires phone verification",
                  [input.vendor.name])
}

findings contains msg if {
    input.vendor.annual_payments >= 600
    not input.vendor.tracking_1099
    msg := sprintf("Vendor '%s' requires 1099 tracking (payments exceed $600)",
                  [input.vendor.name])
}

# Determine if vendor is ready for payment
allow_payment := true if {
    count(findings) == 0
    input.vendor.w9_on_file
    contact_verified
}

```

Policy 3: Expense Documentation

Human Language (for the Policy Manual)

Expense Documentation Policy

All business expenses must be properly documented:

1. Expenses over \$75 require receipt attachment within 10 days of transaction
2. All expenses require a business purpose description
3. Meal expenses require: attendee names, business relationship, and specific purpose
4. Travel expenses require: destination, dates, and business reason
5. Expenses lacking required documentation after 10 days will be flagged for review

Machine Language (for the Policy Engine)

Expense Documentation Policy - OPA Rego format

```
package expense.documentation
```

```
import future.keywords.contains
```

```
import future.keywords.if
```

```
# Calculate days since transaction
```

```
days_since_transaction := result if {
```

```
    result := time.diff(time.now_ns(), input.transaction_date_ns).days
```

```
}
```

```
# Receipt requirements
```

```
receipt_required := true if {
```

```
    input.amount > 75
```

```
}
```

```
# Business purpose requirements
```

```
business_purpose_required := true # Always required
```

```
# Enhanced documentation for meals
```

```
meal_documentation_required := true if {
```

```
    input.category == "meals"
```

```
}
```

```
meal_documentation_required := true if {
```

```
    input.category == "entertainment"
```

```
}
```

```
# Enhanced documentation for travel
```

```
travel_documentation_required := true if {
```

```
    input.category == "travel"
```

```
}
```

```
# Findings
```

```
findings contains msg if {
```

```
    receipt_required
```

```

    not input.has_receipt
    days_since_transaction > 10
    msg := sprintf("Expense $%.2f missing receipt (overdue by %d days)",
                  [input.amount, days_since_transaction - 10])
}

findings contains msg if {
    receipt_required
    not input.has_receipt
    days_since_transaction <= 10
    msg := sprintf("Expense $%.2f requires receipt within %d days",
                  [input.amount, 10 - days_since_transaction])
}

findings contains msg if {
    not input.business_purpose
    msg := "Expense missing business purpose description"
}

findings contains msg if {
    input.business_purpose
    count(input.business_purpose) < 10
    msg := "Business purpose too brief - please provide more detail"
}

findings contains msg if {
    meal_documentation_required
    not input.meal_attendees
    msg := "Meal expense requires attendee names"
}

findings contains msg if {
    meal_documentation_required
    not input.meal_business_relationship
    msg := "Meal expense requires business relationship description"
}

findings contains msg if {
    travel_documentation_required
    not input.travel_destination
    msg := "Travel expense requires destination"
}

findings contains msg if {
    travel_documentation_required
    not input.travel_dates
    msg := "Travel expense requires travel dates"
}

```

```

findings contains msg if {
    travel_documentation_required
    not input.travel_purpose
    msg := "Travel expense requires business purpose"
}

# Severity classification
severity := "critical" if {
    days_since_transaction > 30
    count(findings) > 0
}

severity := "warning" if {
    days_since_transaction > 10
    days_since_transaction <= 30
    count(findings) > 0
}

severity := "info" if {
    days_since_transaction <= 10
    count(findings) > 0
}

```

Policy 4: Bank Reconciliation Currency

Human Language (for the Policy Manual)

Bank Reconciliation Policy

All bank and credit card accounts must be reconciled regularly:

1. Bank accounts must be reconciled within 30 days of statement date
2. Credit card accounts must be reconciled within 15 days of statement date
3. Uncleared items older than 60 days require investigation and documentation
4. The bookkeeper is responsible for reconciliation; the owner reviews monthly
5. Any discrepancy over \$100 requires immediate investigation

Machine Language (for the Policy Engine)

Bank Reconciliation Policy - OPA Rego format

```
package reconciliation.currency
```

```
# Define thresholds by account type
```

```
max_days_bank := 30
```

```
max_days_credit_card := 15
```

```

max_days_uncleared := 60
discrepancy_threshold := 100

# Calculate days since reconciliation
days_since_reconcile := result if {
    result := time.diff(time.now_ns(),
        input.last_reconcile_date_ns).days
}

# Determine reconciliation deadline
reconciliation_deadline := max_days_bank if {
    input.account_type == "bank"
}

reconciliation_deadline := max_days_credit_card if {
    input.account_type == "credit_card"
}

# Check if reconciliation is overdue
reconciliation_overdue := true if {
    days_since_reconcile > reconciliation_deadline
}

# Findings
findings contains msg if {
    reconciliation_overdue
    msg := sprintf("Account '%s' reconciliation overdue by %d days (due: %d days)",
        [input.account_name,
        days_since_reconcile - reconciliation_deadline,
        reconciliation_deadline])
}

findings contains msg if {
    some item in input.uncleared_items
    item.days_outstanding > max_days_uncleared
    msg := sprintf("Uncleared item in '%s' older than %d days: %s ($%.2f)",
        [input.account_name, max_days_uncleared,
        item.description, item.amount])
}

findings contains msg if {
    abs(input.discrepancy) > discrepancy_threshold
    msg := sprintf("Reconciliation discrepancy in '%s' exceeds threshold: $%.2f",
        [input.account_name, input.discrepancy])
}

# Priority scoring
priority := "high" if {

```

```

    days_since_reconcile > reconciliation_deadline + 15
}

priority := "medium" if {
    days_since_reconcile > reconciliation_deadline
    days_since_reconcile <= reconciliation_deadline + 15
}

priority := "low" if {
    days_since_reconcile > reconciliation_deadline - 7
    days_since_reconcile <= reconciliation_deadline
}

```

How to Create Your Own Dual-Language Policies

Step 1: Write the Human Version

- Use clear, plain English
- Define specific thresholds and conditions
- Explain the “why” behind the rule
- Include exceptions and escalation paths

Step 2: Identify the Decision Points

- What values need to be checked?
- What thresholds apply?
- What combinations trigger action?
- What are the possible outcomes?

Step 3: Translate to Machine Language

- Map human conditions to code logic
- Define input variables
- Create findings/alerts for violations
- Add severity and priority where needed

Step 4: Test Both Versions

- Walk through scenarios with the English policy
- Run the same scenarios through the code
- Results should match
- Adjust until aligned

Step 5: Maintain Together

- When human policy changes, update code
- When code reveals edge cases, update human policy
- Review both quarterly for currency

Template: Create Your Own

Policy Name:

Human Version:

[Your policy in plain English]

- 1.
- 2.
- 3.

Decision Points:

Condition	Threshold	Action
-----------	-----------	--------

Machine Version:

```
# Policy Name - format of your choice
```

```
package [namespace]
```

```
# Define thresholds
```

```
# Findings
```

```
# Actions
```

For more resources: projectbits.com/taxready/ch4

Tax Ready Bookkeeping by Don Lovett | ProjectBits Consulting

Tax Ready(TM) Bookkeeping

(c) 2026 ProjectBits Consulting. All rights reserved.