# CICS PROGRAMMING USING COBOL

A Practical Transaction Design & Troubleshooting Workbook

Case study • Pseudocode exercise • Error handling • Test plan

Estimated time: 40–55 minutes | Level: Intermediate | Prerequisite: basic COBOL and JCL familiarity

### What you will practise

Design the flow of an online customer inquiry transaction, distinguish screen interaction from business logic and data access, plan safe error handling, and build tests for VSAM- or DB2-backed processing. This conceptual exercise does not require a live CICS region.

## 1. Scenario: Customer account inquiry

A service representative enters a customer account number on an online screen. The application validates input, retrieves account details, displays a concise result, and handles missing accounts or temporary data-access errors without exposing sensitive information.

| Requirement | Expected behaviour |
|---|---|
| Valid account number | Retrieve the matching record and show only permitted fields. |
| Blank or malformed input | Return a clear validation message; do not issue an unnecessary data request. |
| Account not found | Display a neutral not-found response and preserve a safe path back to the input screen. |
| Data-access failure | Handle the error, log useful diagnostic context according to policy, and avoid showing internal details to the user. |
| Repeated submit | Prevent accidental duplicate actions where relevant; an inquiry should not modify account data. |

### Your task

Learning has never been so easy

Define the expected behaviour for valid input, invalid input, not-found records and data-access failures. Identify what must be tested before the transaction is released.

## 2. Design the transaction flow

**Input screen → validate input → invoke business logic → retrieve record (VSAM or DB2) → map result to response → display result.** At each boundary, define the success path and the invalid-input, not-found, timeout and unexpected-error paths. Keep data access and presentation responsibilities sufficiently separate to test them independently.

### Design choices: pros and cons

| Choice | Pros | Trade-offs / risks |
|---|---|---|
| **Validate before data access** | Avoids unnecessary I/O and gives clearer feedback. | Validation must stay aligned with business rules. |
| **Separate business logic from screen handling** | Improves reuse, testability and clarity. | Requires clear interfaces and disciplined error handling. |
| **Neutral user-facing error** | Avoids exposing internal system or account details. | Support still needs protected diagnostic logs. |
| **Keep inquiry read-only** | Reduces unintended data-change risk. | Any update function needs separate authorization and recovery design. |

## 3. Pseudocode planning exercise

Write language-neutral pseudocode before adding CICS command syntax.

■ Receive the account number from the input screen.



© Copyright protected

■ Validate presence, format and permitted length.

■ If invalid, return a validation message without accessing data.

■ Retrieve the account from the chosen data source.

■ If found, map only approved fields to the response.

■ If not found, return a neutral not-found response.

■ If data access fails, capture approved diagnostic details and return a safe message.

■ Test all paths and verify inquiry processing does not change account data.

## Your pseudocode / notes

_____
_____
_____
_____

ez-learn
Learning has never been so easy


© Copyright protected

## 4. Test plan: prove each path

A happy-path test is not enough. Cover validation, data outcomes, failure paths and response security.

| Test | Input / condition | Expected result |
|---|---|---|
| Valid account | Known test account exists. | Correct permitted fields display; no data change. |
| Blank account | No account number entered. | Validation response; no retrieval attempted. |
| Malformed account | Invalid characters or length. | Validation response; no internal error shown. |
| Unknown account | Well-formed but absent account. | Neutral not-found response; no sensitive leakage. |
| Data source unavailable | Controlled test failure in non-production. | Safe error path; protected diagnostics; no misleading success. |
| Unauthorized user | User lacks required access. | Access denied according to policy; no sensitive fields disclosed. |
| Boundary input | Minimum/maximum allowed lengths. | Correct validation; no truncation or incorrect lookup. |

## 5. How to validate AI-generated design or code suggestions

Treat AI output as a proposal—not proof of correctness. Use only approved tools and sanitised material.

| Validation | Method |
|---|---|
| **Trace to requirements** | Map each suggested rule to an explicit requirement, existing behaviour or approved business rule. |
| **Check technical details** | Verify CICS commands, COBOL syntax, data definitions and VSAM/DB2 access against approved IBM documentation and the target environment. |
| **Review with an SME** | Ask a CICS/COBOL developer to check transaction flow, resource use, security and recovery. |
| **Run independent tests** | Execute tests in a non-production region with controlled test data, including error paths and boundary conditions. |
| **Compare actual results** | Check response, logs, codes, data integrity and unexpected updates against expected outcomes. |
| **Keep evidence** | Record prompt, context, reviewed output, source references, test results, reviewer and unresolved issues. |

### Acceptance rule

Accept the design only when behaviour is traceable to requirements and tests pass in an approved non-production environment. If a requirement, API behaviour or error path is unclear, record it as an open question rather than letting AI fill the gap.


© Copyright protected

## 6. Knowledge check

**Q1.** Why validate input before accessing VSAM or DB2?

**Q2.** Why should user-facing messages avoid internal diagnostics?

**Q3.** What should be tested when an account is not found?

**Q4.** Can AI-generated test cases prove a transaction is correct?

**Q5.** What should you do if actual results differ from expected results?

### Suggested answers

**1.** To reject invalid requests early, reduce unnecessary access and provide consistent feedback.

**2.** Internal details may reveal sensitive information; protected logs can retain support diagnostics.

**3.** The expected neutral response, safe navigation, no sensitive leakage and no unintended data change.

**4.** No. Check them against requirements, review independently and run them in a controlled environment.

**5.** Capture reproducible evidence, investigate code and test data, and keep the defect open until resolved.

## 7. Review checklist

■ Input rules and error paths are defined.

■ Business logic and data access are separated clearly.

■ Only authorised fields are returned.

■ Not-found and data-access failure paths are tested.

■ Read-only inquiry does not alter data.

■ AI suggestions are traceable to requirements and independently tested.

■ Security, logging and operational standards have been reviewed.

### Further learning

Ez-Learn course: CICS Programming using COBOL (VSAM+DB2) — https://ez-learn.global/course/cics-programming. For syntax and product-specific behaviour, consult approved IBM CICS, COBOL, VSAM and Db2 documentation for the target environment.

Educational exercise only. Do not run sample logic against production data. Use your organisation's security, change-control and testing procedures.

Practical takeaway: define transaction behaviour first, test every meaningful path, and validate code suggestions against requirements and evidence.


© Copyright protected