CBACT01C-cbl-0.4.4

Report date: *2025-04-03T01:18:24.986489613Z*

User ID: *c4f80468-40b1-70a8-5e14-14529ceb043e*

AWS Account ID: *689410423121*

This is an AI generated report based on the code carddemo-v2 combined.zip submitted by c4f80468-40b1-70a8-5e14-14529ceb043e. Use of Amazon Q is subject to AWS Responsible AI Policy
https://aws.amazon.com/ai/responsible-ai/policy/

Table of Contents

Documentation

1. High Level Overview Of This Program

1.1. Program Name

CBACT01C

1.2. Main Purpose

The main purpose of CBACT01C is to read account records from an indexed file, process the data, and write the processed information into multiple output files with different formats. This program performs data transformation and distribution across various file types.

1.3. Business Context

This program operates in the context of a card management system, likely for a financial institution. It processes account information, which suggests it's part of a larger system for managing credit card accounts or similar financial products. The program's role appears to be data extraction and reformatting for downstream processes or reporting purposes.

1.4. Key Features

1. File Input/Output: Reads from an indexed file (ACCTFILE-FILE) and writes to three different output files (OUT-FILE, ARRY-FILE, and VBRC-FILE).
2. Data Transformation: Processes account records and reformats the data for different output requirements.
3. Multiple Output Formats: Generates output in fixed-length (OUT-FILE), array-like structure (ARRY-FILE), and variable-length record (VBRC-FILE) formats.
4. Error Handling: Includes file status checks and error reporting for I/O operations.
5. Data Manipulation: Performs calculations and data adjustments, such as setting default values for certain fields (e.g., OUT-ACCT-CURR-CYC-DEBIT).

1.5. Input Output

Input Tables:
N/A

Output Tables:
N/A

Input Files:

1. ACCTFILE-FILE: An indexed file containing account records. This file serves as the primary data source for the program, providing detailed account information for processing.

Output Files:
1. OUT-FILE: A sequential file that stores processed account records in a fixed-length format. This file likely serves as a standardized output for other systems or reporting tools.
2. ARRY-FILE: A sequential file that stores account information in an array-like structure. This file may be used for specialized reporting or analysis that requires multiple balance entries per account.
3. VBRC-FILE: A sequential file with variable-length records. This file stores selected account information in two different record formats, possibly for different downstream processes or systems that require more flexible data structures.

Miscellaneous Input:
N/A

Miscellaneous Output:
1. Console Output: The program displays account record details and processing status messages to the console, which can be used for monitoring and debugging purposes.

1.6. Integration

This program does not directly interact with any external systems. It operates as a standalone data processing unit within the card management system, reading from and writing to files that may be used by other components of the system.

2. Program Logic And Functionality

2.1. Program Logic1

This rule governs the transfer and transformation of account data from an input record to an output record. It ensures that relevant account information is accurately copied and, in some cases, modified before being written to the output file. The rule processes various account details such as the account identifier, active status, current balance, credit limits, important dates, and financial cycle information.

A key aspect of this rule is the special handling of the current cycle debit amount. If this amount is zero in the input record, the rule assigns a default value of 2525.00 to the corresponding field in the output record. This conditional assignment ensures that accounts with no current cycle debit are given a standard value, which may be important for downstream processing or reporting.

The rule also manages the transfer of dates, including the account opening date, expiration date, and reissue date. These dates are crucial for understanding the lifecycle of the account and may trigger other business processes.

By applying this rule, the system maintains data consistency and integrity when moving account information between different stages of processing or different systems. It also allows for the standardization of certain data points, as seen with the current cycle debit amount, which can be essential for accurate financial reporting and account management.

2.2. Program Logic2

This rule populates an array record with account-related information. The rule assigns values to multiple elements of the array, creating a structured record that includes:

1. An account identifier copied from the main account record.
2. Two instances of the current account balance, both set to the same value from the main account record.
3. Two predefined debit amounts for the current cycle, set to fixed values of 1005.00 and 1525.00 respectively.
4. A third set of balance and debit values, both negative, possibly representing a special case or different type of transaction (-1025.00 for balance and -2500.00 for debit).

This structured array allows for storing multiple instances of balance and debit information within a single account record, which could be used for various purposes such as tracking different time periods or transaction types within the same account.

2.3. Program Logic3

This rule is responsible for populating a variable-length record with key account information. Specifically, it assigns two crucial pieces of data to the record:

1. The unique identifier for the account
2. The current status of the account (whether it's active or inactive)

The rule extracts this information from the main account record and prepares it for writing to a separate file. This approach allows for efficient storage and retrieval of essential account details in a format that can accommodate different record lengths. By creating this condensed version of account information, the system can quickly access and process critical account data without needing to read the entire account record each time.

This rule plays a vital role in data management and processing efficiency, as it enables rapid access to key account status information which may be frequently needed for various business operations or reporting purposes.

2.4. Program Logic4

This rule is responsible for populating a variable-length record with key account information. The rule extracts specific data elements from the main account record and assigns them to a condensed format. This condensed record includes:

1. The account identifier, which serves as a unique reference for the account.
2. The current balance of the account, representing its up-to-date financial status.
3. The credit limit assigned to the account, indicating the maximum amount of credit extended.
4. The year when the account or associated card is scheduled for reissue.

By creating this condensed record, the rule facilitates efficient storage and transmission of essential account data. This information can be utilized for various purposes such as reporting, analysis, or further processing. The variable-length nature of the record allows for flexibility in data management while focusing on key account attributes.

3. The Data Flow And Data Dependencies

3.1. Data Flow

1. Input File (ACCTFILE-FILE):
- Data is read from the indexed ACCTFILE-FILE into ACCOUNT-RECORD structure.
- Key data elements: ACCT-ID, ACCT-ACTIVE-STATUS, ACCT-CURR-BAL, ACCT-CREDIT-LIMIT, ACCT-CASH-CREDIT-LIMIT, ACCT-OPEN-DATE, ACCT-EXPIRAION-DATE, ACCT-REISSUE-DATE, ACCT-CURR-CYC-CREDIT, ACCT-CURR-CYC-DEBIT, ACCT-GROUP-ID.

2. Output File (OUT-FILE):
- Data from ACCOUNT-RECORD is transferred to OUT-ACCT-REC structure.
- OUT-ACCT-CURR-CYC-DEBIT is set to 2525.00 if ACCT-CURR-CYC-DEBIT is zero.
- OUT-ACCT-REC is written to OUT-FILE.

3. Array File (ARRY-FILE):
- ARR-ACCT-ID is populated from ACCT-ID.
- ARR-ACCT-CURR-BAL(1) and ARR-ACCT-CURR-BAL(2) are set to ACCT-CURR-BAL.
- ARR-ACCT-CURR-CYC-DEBIT(1) is set to 1005.00.
- ARR-ACCT-CURR-CYC-DEBIT(2) is set to 1525.00.
- ARR-ACCT-CURR-BAL(3) is set to -1025.00.
- ARR-ACCT-CURR-CYC-DEBIT(3) is set to -2500.00.
- ARR-ARRAY-REC is written to ARRY-FILE.

4. Variable-length Record File (VBRC-FILE):
- Two types of records are written to VBRC-FILE:
a. VBRC-REC1: Contains VB1-ACCT-ID and VB1-ACCT-ACTIVE-STATUS.
b. VBRC-REC2: Contains VB2-ACCT-ID, VB2-ACCT-CURR-BAL, VB2-ACCT-CREDIT-LIMIT, and VB2-ACCT-REISSUE-YYYY.
- Data is populated from corresponding fields in ACCOUNT-RECORD.
- Records are written with varying lengths (12 bytes for VBRC-REC1 and 39 bytes for VBRC-REC2).

3.2. Data Dependency

1. ACCOUNT-RECORD Structure:
- This structure is the primary source of data for all output operations.
- Fields in ACCOUNT-RECORD directly correspond to fields in OUT-ACCT-REC, ARR-ARRAY-REC, VBRC-REC1, and VBRC-REC2.

2. OUT-ACCT-CURR-CYC-DEBIT Calculation:
- Depends on ACCT-CURR-CYC-DEBIT from ACCOUNT-RECORD.
- If ACCT-CURR-CYC-DEBIT is zero, OUT-ACCT-CURR-CYC-DEBIT is set to 2525.00.

3. ARR-ARRAY-REC Population:
- ARR-ACCT-CURR-BAL(1) and ARR-ACCT-CURR-BAL(2) depend on ACCT-CURR-BAL.
- Other fields (ARR-ACCT-CURR-CYC-DEBIT(1), ARR-ACCT-CURR-CYC-DEBIT(2), ARR-ACCT-CURR-BAL(3), ARR-ACCT-CURR-CYC-DEBIT(3)) are populated with hard-coded values.

4. VBRC-REC2 Reissue Date:
- VB2-ACCT-REISSUE-YYYY depends on WS-ACCT-REISSUE-YYYY, which is derived from ACCT-REISSUE-DATE in ACCOUNT-RECORD.

5. Record Length for Variable-length Records:
- WS-RECD-LEN determines the length of records written to VBRC-FILE.
- Set to 12 for VBRC-REC1 and 39 for VBRC-REC2.

6. Error Handling:
- APPL-RESULT depends on the status of file operations (ACCTFILE-STATUS, OUTFILE-STATUS, ARRYFILE-STATUS, VBRCFILE-STATUS).
- END-OF-FILE flag depends on APPL-RESULT when it indicates end-of-file condition (APPL-EOF).

7. IO-STATUS Display:
- IO-STATUS-04 depends on IO-STATUS for error reporting.
- TWO-BYTES-BINARY is used to convert IO-STAT2 to a displayable format in IO-STATUS-0403.

4. Database Information

4.1. Database Overview

N/A

4.2. Database Connectivity

N/A

4.3. Database Schema

N/A

4.4. SQLQueries And Commands

N/A

4.5. Data Flow

N/A

4.6. Error Handling And Transaction Management

N/A

4.7. Additional Considerations

While this COBOL program (CBACT01C) does not directly interact with a database, it does perform file operations that are worth noting:

1. File Reading:
- The program reads from an indexed file named ACCTFILE-FILE sequentially.
- The file contains account records with various fields such as account ID, balance, credit limit, etc.

2. File Writing:
- The program writes to three output files:
a. OUT-FILE: A sequential file containing processed account records.
b. ARRY-FILE: A sequential file containing array-based account records.
c. VBRC-FILE: A variable-length record file containing two types of records.

3. Error Handling:
- The program checks file status codes after each file operation.

- If an error occurs during file operations, the program displays an error message, shows the file status, and abends the program.

4. Data Processing:
- The program reads account records, processes them, and writes the data to different output files in various formats.
- It performs data transformations, such as populating array records and creating variable-length records.

5. File Status Checking:
- The program consistently checks file status codes (e.g., ACCTFILE-STATUS, OUTFILE-STATUS, ARRYFILE-STATUS, VBRCFILE-STATUS) to ensure successful file operations.

6. End-of-File Handling:
- The program uses an END-OF-FILE flag to control the main processing loop, ensuring all records are processed.

While not database-related, these file operations and data handling practices are crucial for the program's functionality and data integrity.

5. Error And Abend Handling

5.1. Overall Strategy

The program CBACT01C.cbl employs a consistent error handling strategy throughout its execution. It primarily relies on checking file status codes after file operations and uses conditional statements to detect and respond to errors. When an error is encountered, the program typically displays an error message, logs the error details, and then abends the program using a call to 'CEE3ABD'.

5.2. Common Error Conditions

5.2.1. List Errors

1. File opening errors for ACCTFILE, OUTFILE, ARRYFILE, and VBRCFILE
2. File reading errors for ACCTFILE
3. File writing errors for OUTFILE, ARRYFILE, and VBRCFILE
4. File closing errors for ACCTFILE

5.2.2. Error Sources

1. File opening errors: Could be due to missing files, incorrect file permissions, or system resource issues.
2. File reading errors: Possibly caused by corrupted data, unexpected end-of-file, or hardware issues.
3. File writing errors: May occur due to insufficient disk space, file locking issues, or hardware failures.
4. File closing errors: Potentially caused by system resource issues or unexpected program termination.

5.3. Error Detection Mechanisms

5.3.1. Detect Methods

The program uses file status checking after each file operation. It checks the status codes stored in variables like ACCTFILE-STATUS, OUTFILE-STATUS, ARRYFILE-STATUS, and VBRCFILE-STATUS. These status codes are compared against expected values ('00' for success, '10' for end-of-file) to detect errors.

5.3.2. Functions And Exception Check

The program uses IF-ELSE statements to check file status codes. For example:
- In the file opening routines (0000-ACCTFILE-OPEN, 2000-OUTFILE-OPEN, etc.), it checks if the status is '00'.
- In the 1000-ACCTFILE-GET-NEXT paragraph, it checks for '00' (success), '10' (EOF), and other status codes.

- In the write routines (1350-WRITE-ACCT-RECORD, 1450-WRITE-ARRY-RECORD, etc.), it checks if the status is not '00' and not '10'.

5.4. Error Responses And Recovery Actions

5.4.1. Immediate Response

When an error is detected, the program typically:
1. Displays an error message indicating the nature of the error (e.g., "ERROR OPENING ACCTFILE").
2. Moves the file status to a common IO-STATUS variable.
3. Performs the 9910-DISPLAY-IO-STATUS paragraph to log detailed error information.
4. Calls the 9999-ABEND-PROGRAM paragraph to terminate the program.

5.4.2. Recovery Procedures

The program does not implement specific recovery procedures or retry mechanisms. Upon encountering an error, it opts for immediate termination through the 9999-ABEND-PROGRAM paragraph. This paragraph:
1. Displays an "ABENDING PROGRAM" message.
2. Sets an ABCODE of 999.
3. Calls 'CEE3ABD' to abnormally terminate the program.

5.5. Error Logging Mechanisms

5.5.1. Mechanisms Details

The program uses the 9910-DISPLAY-IO-STATUS paragraph for error logging. This paragraph:
1. Checks if the IO-STATUS is numeric or if IO-STAT1 is '9'.
2. Formats the error status into a 4-digit code (NNNN format).
3. Displays the formatted error status using the DISPLAY statement.

5.5.2. Error Logs Format And Content

The error logs are displayed on the console in the format:
"FILE STATUS IS: NNNN" where NNNN is the formatted error status code.
The program does not implement a separate error log file or use a dedicated logging system. All error messages are output directly to the console.

5.6. User Notifications And Interface Handling

5.6.1. Errors Communication

Errors are communicated through console messages. The program uses DISPLAY statements to output error messages, such as:
- "ERROR OPENING ACCTFILE"
- "ERROR READING ACCOUNT FILE"
- "ACCOUNT FILE WRITE STATUS IS: [status]"

These messages are intended for system operators or developers monitoring the program's execution.

5.6.2. Error Messages Interface

The program does not implement a user interface for displaying error messages. All error notifications are console-based, suitable for batch processing environments. There are no specific instructions or guidance provided to users for resolving issues; the program terminates upon encountering errors.

6. The Input And Output Processing

6.1. Input Requirements

6.1.1. Input Data Type And Significance

The main input for this program is the ACCTFILE, which contains account records. This input is significant as it provides the account information that will be processed and written to various output files.

6.1.2. Input Source

The input source is an indexed file named ACCTFILE, which is likely an external system or database containing account information.

6.1.3. Input Format And Structure

6.1.3.1. Field Lengths

- FD-ACCT-ID: 11 digits
- FD-ACCT-DATA: 289 characters

6.1.3.2. Mandatory Fields

- FD-ACCT-ID (as it is the record key for the indexed file)

6.1.3.3. Data Types

- FD-ACCT-ID: Numeric (PIC 9(11))
- FD-ACCT-DATA: Alphanumeric (PIC X(289))

6.1.3.4. Additional Requirements

The input file is organized as an indexed file with sequential access mode, using FD-ACCT-ID as the record key.

6.2. Output Specifications

6.2.1. Output Purpose And Downstream Usage

The program generates three output files:
1. OUTFILE: Contains processed account records.
2. ARRYFILE: Contains array-based account information.
3. VBRCFILE: Contains variable-length records with account information.
These outputs are likely used by downstream processes for further analysis, reporting, or integration with other systems.

6.2.2. Output Destination

All three output files (OUTFILE, ARRYFILE, and VBRCFILE) are written to external sequential files, which could be used by other programs, systems, or databases for further processing.

6.2.3. Output Format And Structure

6.2.3.1. Formatting Rules

- OUTFILE: Fixed-length records with specific fields for account information.
- ARRYFILE: Fixed-length records with an array structure for account balances and cycle debits.
- VBRCFILE: Variable-length records with two different formats (VB1 and VB2).

6.2.3.2. Structure

- OUTFILE: Sequential file with fixed-length records (OUT-ACCT-REC).
- ARRYFILE: Sequential file with fixed-length records (ARR-ARRAY-REC).
- VBRCFILE: Sequential file with variable-length records (VBR-REC), length varying from 10 to 80 characters.

6.3. Error Handling

6.3.1. Input Data Errors Handling

The program checks the ACCTFILE-STATUS after each READ operation. If an error occurs (status not '00' or '10'), it displays an error message, shows the I/O status, and abends the program using the 9999-ABEND-PROGRAM paragraph.

6.3.2. Output Errors Handling

For each WRITE operation to OUTFILE, ARRYFILE, and VBRCFILE, the program checks the respective file status. If an error occurs (status not '00' and not '10'), it displays an error message, shows the I/O status, and abends the program using the 9999-ABEND-PROGRAM paragraph.

7. Integration Information

7.1. Integration Overview

7.1.1. Purpose Of Integration

N/A

7.1.2. Integration Interactions

N/A

7.2. Integration Type

N/A

7.3. Interacted External Systems

N/A

7.4. External System Integration

7.4.1. System Type

N/A

7.4.2. Tech Stack

N/A

7.4.3. Integration Role Of External System

N/A

7.5. External System Data Flow

N/A

7.6. External System Interface

N/A

7.7. External System Errors Handling

N/A

8. Screen Documentation

This program does not have any screen.

9. Programs Called

9.1. Application Programs

N/A

9.2. System Programs

CEE3ABD, OUTFILE

9.3. Missing Programs

N/A