

1. Program Overview

Table of Contents

1. Program Overview
2. Program Architecture
3. Program Structure
4. Data Management
5. Control Flow
6. Error Handling
7. Integration & Interfaces

Section 1: Overview and Introduction

Purpose and Main Functionality

The program `P3BFZ01` is a PL/I-based application designed to process financial and material movement data within the *Hausteile-Buchungssystem* (Internal Parts Booking System). Its primary purpose is to evaluate and transform input data from various sources, such as financial records and material movement data, into structured outputs for reporting, accounting, and further downstream processing. The program achieves this through a modular design that includes file operations, conditional logic, and subprogram calls.

Key functionalities include:

1. **Data Transformation:** The program transforms input structures like `BEBU` into output structures such as `IN` and `AT`. For example, the procedure `UMSETZEN_IN` maps fields from `BEBU` to `IN` while handling specific business rules for scrap (`BEBU.UEBGEB = 'AUSSCHUSS'`).
2. **Cost Evaluation:** Procedures like `BEWERTEN` and `ERMITTELN_IN_E_PRS` calculate costs and assign values to fields such as `E_PRS_FW` and `E_PRS_ABW` based on input quantities (`MENGE`).
3. **File Operations:** The program reads from and writes to multiple files, such as `BFKARTE`, `BFBUOUT`, and `BFLISTE`. For instance, the procedure `WRITE_BFBUOUT` writes transformed data from the `AT` structure to the `BFBUOUT` file.
4. **Error Handling:** The program includes mechanisms to handle errors during file operations, database connections, and subprogram calls. For example, the procedure `U_P9AT003` signals an error if no valid billing period is found.

Business Context and Importance

The program operates within the Production Flow Accounting domain, where it plays a critical role in managing the financial and material data flow for internal parts booking. This system is essential for ensuring accurate cost allocation, inventory management, and compliance with financial reporting standards.

Key Business Rules

1. Scrap Handling: When processing scrap (`BEBU.UEBGEB = 'AUSSCHUSS'`), the program adjusts values such as `IN.G_PRS_FW` and `IN.MENGE` and writes the results to the `ISTKOSIN` file.
2. Account Adjustments: Specific account numbers are updated based on predefined rules. For example, in `UMSETZEN_BF`, if `BEBU.HW = '01'` and `BEBU.KTO = '6000'`, the account is updated to `'6001'`.
3. Billing Period Determination: The procedure `U_P9AT003` calls the subprogram `P9AT003` to determine the current billing period from a characteristic table database.

The program's outputs are used by downstream systems for financial reporting, cost analysis, and decision-making. Its ability to handle complex business rules ensures data integrity and compliance with organizational policies.

Key Stakeholders and Users

Stakeholders

1. Finance Department: Uses the program's outputs for cost allocation, financial reporting, and compliance.
2. Production Management: Relies on material movement data for inventory tracking and production planning.
3. IT Operations: Maintains the program and ensures its integration with other systems.

Users

1. Financial Analysts: Use reports generated by the program to analyze costs and profitability.
2. System Administrators: Monitor the program's performance and handle error logs.
3. Developers: Extend or modify the program to accommodate new business requirements.

System Requirements

Hardware Requirements

- Processor: Minimum 2.4 GHz quad-core CPU.
- Memory: At least 8 GB of RAM.
- Storage: 500 MB of free disk space for program files and temporary data.

Software Requirements

- Operating System: IBM z/OS or equivalent mainframe environment.
- Compiler: IBM PL/I compiler.
- Database: IBM DB2 for database operations.

- File System: Support for fixed-block files, as used in BFKARTE and BFBUOUT.

Technical Specifications

1. Programming Language: PL/I.

2. Input Structures:

- BEBU: Contains fields like MENGE, KTO, UEBGEB, and HK_PROP.
- WKARTE: Includes program metadata and test flags.

5. Output Structures:

- IN: Transformed structure for material movement data.
- AT: Contains financial data for output to BFBUOUT.

8. Files:

- Input: BFKARTE, BFSFIN.
- Output: BFBUOUT, BFLISTE, ISTKOSIN.

Example Implementation

The procedure UMSETZEN_IN demonstrates the program's technical complexity:

```
UMSETZEN_IN: PROC REORDER;
  IF BEBU.UEBGEB = 'AUSSCHUSS' THEN DO;
    IN.G_PRS_FW = ROUND(BEBU.HK_PROP / BEBU.MENGE, 3);
    IN.MENGE = BEBU.MENGE * (-1);
    CALL WRITE_ISTKOSIN;
  END;
  ELSE DO;
    IN.G_PRS_FW = BEBU.HK_PROP;
    IN.MENGE = BEBU.MENGE;
  END;
END UMSETZEN_IN;
```

Performance Considerations

1. Data Volume: The program processes large volumes of financial and material data. Efficient file I/O operations are critical for performance.
2. Error Handling: Robust error-handling mechanisms, such as those in U_P9AT003, ensure minimal downtime and quick recovery.
3. Scalability: The modular design allows for easy scaling to accommodate additional business rules or increased data volumes.
4. Optimization: Procedures like ERMITTELN_IN_E_PRS use rounding and conditional logic to minimize computational overhead.

Example Optimization

The procedure ERMITTELN_IN_E_PRS optimizes calculations by avoiding division when MENGE is zero:

```
IF IN.MENGE = 0 THEN DO;
```

```

    IN.E_PRS_FW = IN.G_PRS_FW;
    IN.E_PRS_ABW = IN.G_PRS_ABW;
END;
ELSE DO;
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW / IN.MENGE, 3);
    IN.E_PRS_ABW = ROUND(IN.G_PRS_ABW / IN.MENGE, 3);
END;

```

By adhering to these considerations, the program ensures reliable and efficient processing of critical financial and material data.

2. Program Architecture

Section 2: High-Level System Design and Integration

High-Level System Design

The program **P3BFZ01** is a modular PL/I application designed to process financial and material movement data within the *Hausteile-Buchungssystem* (Internal Parts Booking System). It transforms input data into structured outputs for reporting, accounting, and downstream processing. The program is structured into distinct procedures, each responsible for specific tasks such as data transformation, cost evaluation, and file operations. The modular design ensures maintainability and scalability.

Core Components

1. Main Procedure (**P3BFZ01**):

- Declared with `OPTIONS (MAIN)` and serves as the entry point for the program.
- Accepts a single parameter, `QCB1`, which is passed to subprocedures like `U_P9AT003` for database interactions.

4. Data Transformation Procedures:

- **UMSETZEN_IN**: Maps fields from the `BEBU` structure to the `IN` structure. For example:

```

IN.MENGE = BEBU.MENGE;
IN.G_PRS_FW = BEBU.HK_PROP;

```

Handles special cases such as scrap (`BEBU.UEBGEB = 'AUSSCHUSS'`), where `IN.MENGE` is adjusted to `BEBU.MENGE * (-1)`.

- **UMSETZEN_BF**: Updates financial account fields in the `BEBU` structure based on specific conditions:

```

IF BEBU.HW = '01' AND BEBU.KTO = '6000' THEN BEBU.KTO = '6001';

```

8. Cost Evaluation Procedures:

- **BEWERTEN:** Evaluates costs based on input parameters and updates financial data.
- **ERMITTELN_IN_E_PRS:** Calculates unit prices (E_PRS_FW and E_PRS_ABW) based on the quantity (MENGE):

```
IF IN.MENGE = 0 THEN DO;
    IN.E_PRS_FW = IN.G_PRS_FW;
    IN.E_PRS_ABW = IN.G_PRS_ABW;
END;
ELSE DO;
    IN.E_PRS_FW = ROUND (IN.G_PRS_FW / IN.MENGE, 3);
    IN.E_PRS_ABW = ROUND (IN.G_PRS_ABW / IN.MENGE, 3);
END;
```

11. File Operations:

- **WRITE_BFBUOUT:** Writes transformed data from the AT structure to the BFBUOUT file. For example:

```
WRITE FILE (BFBUOUT) FROM (AT);
```

13. Database Interaction:

- **U_P9AT003:** Calls the subprogram P9AT003 to determine the current billing period from a characteristic table database. The results are stored in P9AT003_S:

```
CALL P9AT003 (ADDR (P9AT003_S));
```

Integration Points with Other Systems

The program integrates with multiple external systems and databases to retrieve input data, process it, and generate outputs.

Key Integration Points

1. Database Interaction:

- The program interacts with the DB2 database to retrieve billing period information using the subprogram P9AT003. The SQLCA structure is included for SQL communication:

```
EXEC SQL INCLUDE SQLCA;
```

3. File-Based Integration:

- Input files such as BFSFIN and BFKARTE provide financial and control card data, respectively.
- Output files such as BFBUOUT and BFLISTE are used for financial records and reporting.

6. Subprogram Calls:

- External subprograms like P9BF150 and P9BF105 are used for cost evaluation and account determination:

```
CALL P9BF150 (ADDR(P9BF150_S));
```

8. ISTKOSIN Integration:

- The program writes cost data to the ISTKOSIN system using the procedure `WRITE_ISTKOSIN`.
-

Data Flow Architecture

The data flow in P3BFZ01 follows a structured pipeline:

1. Input Data:

- Financial data is read from `BFSFIN` into the `BEBU` structure.
- Control card data is read from `BFKARTE` into the `WKARTE` structure.

4. Processing:

- Data is transformed and evaluated through procedures like `UMSETZEN_IN` and `BEWERTEN`.
- Special cases, such as scrap handling, are processed conditionally.

7. Output Data:

- Transformed data is written to output files (`BFBUOUT`, `BFLISTE`) or passed to external systems (`ISTKOSIN`).

Example Data Flow

- Input: `BEBU.MENGE = 100`, `BEBU.HK_PROP = 5000`
- Processing:

```
IN.MENGE = BEBU.MENGE;  
IN.G_PRS_FW = BEBU.HK_PROP / BEBU.MENGE; /* Result: 50 */
```

- Output: `IN.G_PRS_FW = 50` written to `ISTKOSIN`.
-

Security Considerations

1. Database Security:

- The program uses the `SQLCA` structure for secure SQL communication. However, no explicit encryption or authentication mechanisms are mentioned in the code.

3. File Access:

- Input and output files are accessed directly. Proper file permissions must be enforced to prevent unauthorized access.

5. Error Handling:

- Procedures like U_P9AT003 include basic error handling:

```
IF P9AT003_S.RETURN_CODE ^= '00' THEN SIGNAL ERROR;
```

7. Data Validation:

- The program assumes valid input data. Additional validation mechanisms should be implemented to prevent processing invalid or malicious data.

System Boundaries

The program operates within the following boundaries:

1. Input Sources:

- Files: BFSFIN, BFKARTE
- Database: DB2 (via subprograms like P9AT003)

4. Output Destinations:

- Files: BFBUOUT, BFLISTE, ISTKOSIN

6. External Dependencies:

- Subprograms: P9BF150, P9BF105, P9AT003

External Interfaces

1. File Interfaces:

- Input:

BFSFIN: Fixed block size of 800.

BFKARTE: Fixed block size of 80.

- Output:

BFBUOUT: Financial records.

BFLISTE: Reports.

8. Database Interfaces:

- Subprogram P9AT003 interacts with the DB2 database to retrieve billing period information.

10. Subprogram Interfaces:

- P9BF150: Evaluates costs.
- P9BF105: Determines manufacturing cost accounts.

Technical Dependencies

1. Programming Language:

- PL/I with features like `PROC REORDER` for optimized execution.

3. Database:

- IBM DB2 for SQL operations.

5. File System:

- Fixed-block file support for input and output files.

7. External Libraries:

- Include files such as `SQLCA`, `P9BF15FI`, and `BFSTD01I` provide necessary structures and macros.

9. Error Handling:

- Basic mechanisms like `SIGNAL ERROR` and `ON ERROR` blocks are used.

This section provides a comprehensive overview of the system's design, integration points, data flow, and dependencies, ensuring a clear understanding of its architecture and functionality. # 3. Program Structure

Section 3: Procedures and Functions

This section provides a detailed analysis of the procedures and functions in the program `P3BFZ01`. It includes their specific purposes, input/output parameters, validation rules, error handling, business rules implementation, performance considerations, and dependencies. Additionally, a call hierarchy is presented to illustrate the relationships and execution paths between procedures.

A. Procedures and Functions

1. Main Procedure: **P3BFZ01**

- Purpose: Serves as the entry point for the program. It initializes variables, opens files, and orchestrates the execution of other procedures.
- Input Parameters:

`QCB1`: A parameter passed to subprocedures like `U_P9AT003` for database interactions.

- Output Parameters: None explicitly defined.
- Validation Rules:

Ensures that all required files are opened successfully.

Validates the `QCB1` parameter before passing it to subprocedures.

- Error Handling:

Uses `ON ERROR` blocks to capture runtime errors.
Calls `PLIDUMP` for detailed error diagnostics.

- Business Rules:

Executes specific logic for financial and material movement data processing.
Handles control card data (`BFKARTE`) to determine program behavior.

- Performance Considerations:

Uses the `REORDER` option to optimize execution.
Minimizes file I/O operations by batching reads and writes.

- Dependencies:

Calls subprocedures like `U_P9AT003`, `UMSETZEN_IN`, and `WRITE_BFBUOUT`.

2. Procedure: **U_P9AT003**

- Purpose: Determines the current billing period by calling the subprogram `P9AT003` and processes the results.
- Input Parameters:

`P9AT003_S`: A structure containing fields like `RETURN_CODE`, `EIN.PR_STAT`, and `PTR_KR17U`.

- Output Parameters:

Updates `P9AT003_S` with the billing date and period.
Outputs the billing period (`ABR_PER`) in the format `YYYYMM`.

- Validation Rules:

Ensures `P9AT003_S.RETURN_CODE` is `'00'` before processing results.

- Error Handling:

Logs an error message if no valid billing period is found.
Signals an error using `SIGNAL ERROR`.

- Business Rules:

Retrieves the billing period from the characteristic table database.
Converts the billing date into a standardized format.

- Performance Considerations:

Minimizes database calls by caching results in `P9AT003_S`.

- Dependencies:

Calls the subprogram `P9AT003`.

3. Procedure: **UMSETZEN_IN**

- Purpose: Transforms the `BEBU` structure into the `IN` structure and processes specific cases based on `BEBU.UEBGEB`.
- Input Parameters:

`BEBU`: A structure containing financial data fields like `MENGE`, `HK_PROP`, and `UEBGEB`.

- Output Parameters:

`IN`: A structure populated with transformed data.

- Validation Rules:

Ensures `BEBU.MENGE` is valid before performing calculations.

- Error Handling:

Handles invalid or missing values in `BEBU` by assigning default values to `IN`.

- Business Rules:

Adjusts `IN.MENGE` for scrap (`BEBU.UEBGEB = 'AUSSCHUSS'`).
Calculates unit prices (`E_PRS_FW` and `E_PRS_ABW`) based on `BEBU.MENGE`.

- Performance Considerations:

Uses conditional logic to minimize unnecessary calculations.

- Dependencies:

Calls `ERMITTELN_IN_E_PRS` for price calculations.
Calls `WRITE_ISTKOSIN` to write processed data.

4. Procedure: **WRITE_BFBUOUT**

- Purpose: Writes transformed data from the `AT` structure to the `BFBUOUT` file.
- Input Parameters:

`AT`: A structure containing fields like `KT_UEBGEB`, `KT_KOAKTO`, and `KT_HKTO`.

- Output Parameters:

Writes the `AT` structure to the `BFBUOUT` file.

- Validation Rules:

Ensures `AT.KT_UEBGEB` is valid before modifying `KT_KOAKTO` and `KT_HKTO`.

- Error Handling:

Does not explicitly handle file write errors.

- Business Rules:

Modifies account fields (KT_KOAKTO and KT_HKTO) for scrap records.

- Performance Considerations:

Batches file writes to reduce I/O overhead.

- Dependencies:

None explicitly mentioned.

5. Procedure: **ERMITTELN_IN_E_PRS**

- Purpose: Calculates unit prices (E_PRS_FW and E_PRS_ABW) based on the quantity (MENGE).
- Input Parameters:

IN.MENGE, IN.G_PRS_FW, IN.G_PRS_ABW.

- Output Parameters:

IN.E_PRS_FW, IN.E_PRS_ABW.

- Validation Rules:

Ensures IN.MENGE is not zero before performing division.

- Error Handling:

Assigns default values if IN.MENGE is zero.

- Business Rules:

Rounds calculated prices to three decimal places.

- Performance Considerations:

Uses efficient arithmetic operations to minimize computation time.

- Dependencies:

None explicitly mentioned.

6. Procedure: **DUPL_GUSS_KO**

- Purpose: Duplicates and processes a record in the GUSSKO structure, writes it to the BFALL file, and updates counters.
- Input Parameters:

GUSSKO: A structure containing fields like MATKO and HK_PROP.

- Output Parameters:

Writes the modified GUSSKO structure to the BFALL file.

Updates counters WANZ (8) and WANZ (6) .

- Validation Rules:

Ensures GUSSKO fields are initialized before processing.

- Error Handling:

Does not explicitly handle errors.

- Business Rules:

Adjusts MATKO and HK_PROP for casting costs.

- Performance Considerations:

Minimizes redundant calculations by reusing values.

- Dependencies:

Calls UMSETZEN_BU to transform the GUSSKO structure.

B. Call Hierarchy

Call Tree

The following diagram illustrates the relationships and execution paths between procedures:

```
P3BFZ01
  U_P9AT003
    P9AT003 (Subprogram)
  UMSETZEN_IN
    ERMITTELN_IN_E_PRS
    WRITE_ISTKOSIN
  WRITE_BFBUOUT
  DUPL_GUSS_KO
    UMSETZEN_BU
  ERMITTELN_IN_E_PRS
```

Execution Paths

1. Main Execution Path:

- P3BFZ01 initializes the program, reads input files, and calls UMSETZEN_IN to process records.
- Transformed data is written to BFBUOUT using WRITE_BFBUOUT.

4. Conditional Execution:

- UMSETZEN_IN calls ERMITTELN_IN_E_PRS only if BEBU.MENGE is not zero.
- DUPL_GUSS_KO is executed only for specific conditions related to casting costs.

7. Recursive Relationships:

- No recursive relationships are present in the program.

9. Module Dependencies:

- The program depends on external subprograms like P9AT003 for database interactions.
- File operations rely on predefined structures like BEBU and AT.

Example Execution Path

For a record with `BEBU.UEBGEB = 'AUSSCHUSS'`:

1. `UMSETZEN_IN` adjusts `IN.MENGE` and calls `ERMITTELN_IN_E_PRS` to calculate unit prices.
2. The transformed `IN` structure is passed to `WRITE_ISTKOSIN` for output.
3. If the record is a scrap record, `WRITE_BFBUOUT` modifies account fields before writing to `BFBUOUT`.

This section provides a comprehensive overview of the program's procedures and their interactions, ensuring a clear understanding of its functionality and execution flow. # 4. Data Management

Section 4: Data Structures, File Operations, and Data Flow

This section provides a detailed analysis of the data structures, file operations, and data flow in the program P3BFZ01. It includes specific examples from the code, exact procedure names and parameters, and implementation details to ensure clarity and completeness.

A. Data Structures

1. Structure: **BEBU**

- Purpose: Represents financial data used throughout the program. It serves as the primary input structure for various procedures and is transformed into other structures like `IN` and `MPS`.
- Fields:

`MANDANT (CHAR(3))`: Represents the client identifier.

`BUCH_WAEHR (CHAR(3))`: Indicates the booking currency.

`MENGE (DECIMAL(15,3))`: Represents the quantity of material or financial units.

`UEBGEB (CHAR(2))`: Indicates the transition area or type of record (e.g., scrap, invoice).

`HK_PROP (DECIMAL(15,2))`: Represents the proportional manufacturing costs.

`KTO (CHAR(4))`: Represents the account number.

`KOAKTO (CHAR(4))`: Represents the offset account number.

- Data Types and Constraints:

MENGE must be greater than or equal to zero.
UEBGEB must match predefined values (e.g., 'AUSSCHUSS' for scrap).
KTO and KOAKTO must be valid account numbers starting with specific prefixes (e.g., '4' for scrap accounts).

- Validation Rules:

MENGE is validated in procedures like ERMITTELN_IN_E_PRS to ensure it is not zero before performing calculations.
UEBGEB is validated in UMSETZEN_IN to determine specific processing logic.

- Usage Patterns:

Passed as an input to procedures like UMSETZEN_IN, UMSETZEN_BF, and DUPL_GUSS_KO.
Transformed into the IN structure in UMSETZEN_IN.

- Relationships:

BEBU is closely related to IN, MPS, and GUSSKO structures, as it serves as the source for their data.

2. Structure: **IN**

- Purpose: Represents transformed financial data used for further processing and output.
- Fields:

MENGE (DECIMAL(15,3)): Transformed quantity from BEBU.MENGE.
G_PRS_FW (DECIMAL(15,2)): Represents the gross price in the local currency.
E_PRS_FW (DECIMAL(15,3)): Represents the calculated unit price in the local currency.
UEBGEB (CHAR(2)): Transition area, directly mapped from BEBU.UEBGEB.

- Data Types and Constraints:

MENGE must be greater than zero for price calculations.
E_PRS_FW is calculated as $\text{ROUND}(G_PRS_FW / MENGE, 3)$.

- Validation Rules:

MENGE is validated in ERMITTELN_IN_E_PRS to ensure it is not zero before performing division.

- Usage Patterns:

Populated in UMSETZEN_IN by mapping fields from BEBU.
Used as input to WRITE_ISTKOSIN for writing to output files.

- Relationships:

Derived from BEBU and used as input for output-related procedures.

3. Structure: **GUSSKO**

- Purpose: Represents casting cost data for scrap records.
- Fields:

KZ_GUSS (CHAR(1)): Indicates casting cost records ('G').

MATKO (DECIMAL(15,2)): Represents material costs for casting.

HK_PROP (DECIMAL(15,2)): Represents proportional manufacturing costs for casting.

- Data Types and Constraints:

MATKO and HK_PROP are calculated as $\text{MATKO_GUSS_SOLL} * (-1)$.

- Validation Rules:

MATKO and HK_PROP must be negative for scrap records.

- Usage Patterns:

Populated in DUPL_GUSS_KO and written to the BFALL file.

- Relationships:

Derived from BEBU and processed independently for casting costs.

4. Structure: **P9AT003_S**

- Purpose: Used for determining the billing period by interacting with the P9AT003 subprogram.
- Fields:

RETURN_CODE (CHAR(2)): Indicates the success or failure of the subprogram call.

EIN.PR_STAT (CHAR(1)): Input parameter for the subprogram, set to '1'.

AUS.\$JAHR, AUS.\$MONAT, AUS.\$TAG (CHAR(4), CHAR(2), CHAR(2)): Represent the billing date (year, month, day).

- Data Types and Constraints:

RETURN_CODE must be '00' for successful processing.

- Validation Rules:

RETURN_CODE is validated in U_P9AT003 to ensure successful execution of the subprogram.

- Usage Patterns:

Passed as input to P9AT003 and updated with the billing period details.

- Relationships:

Used exclusively in U_P9AT003 for billing period determination.

B. File Operations

1. Input Files

- File: **BFSFIN**

Purpose: Contains financial records for processing.

Format: Fixed block size of 800.

Structure: Records are read into the BEBU structure.

Handling:

Read in the main program loop (DO WHILE SATZ_VORH).

Validated for end-of-file using ON ENDFILE.

Error Handling:

Errors during file read operations are captured using ON ERROR and logged with PLIDUMP.

- File: **BFKARTE**

Purpose: Contains control card data for program configuration.

Format: Fixed block size of 80.

Structure: Records are read into the WKARTE structure.

Handling:

Read in U_LESEN_BFKARTE.

Error Handling:

Errors during file read operations are logged.

2. Output Files

- File: **BFBUOUT**

Purpose: Stores processed financial records.

Format: Fixed block size.

Structure: Records are written from the AT structure.

Handling:

Written in WRITE_BFBUOUT.

Modifies account fields (KT_KOAKTO, KT_HKTO) for scrap records.

Error Handling:

No explicit error handling for write operations.

- File: **BFALL**

Purpose: Stores all processed records, including casting costs.

Format: Fixed block size.

Structure: Records are written from the GUSSKO structure.

Handling:

Written in `DUPL_GUSS_KO`.

Error Handling:

No explicit error handling for write operations.

C. Data Flow

1. Data Transformation Logic

- Example: **UMSETZEN_IN**

Transforms `BEBU` into `IN` by mapping fields and performing calculations.
Handles specific cases based on `BEBU.UEBGEB`:

For scrap (`AUSSCHUSS`), adjusts `IN.MENGE` and calls `ERMITTELN_IN_E_PRS`.

Writes the transformed `IN` structure to output using `WRITE_ISTKOSIN`.

- Example: **DUPL_GUSS_KO**

Creates a duplicate record for casting costs in `GUSSKO`.
Modifies fields (`MATKO`, `HK_PROP`) and writes to `BFALL`.

2. Validation Processes

- Example: **ERMITTELN_IN_E_PRS**

Validates `IN.MENGE` to ensure it is not zero before performing division.
Assigns default values if validation fails.

- Example: **U_P9AT003**

Validates `P9AT003_S.RETURN_CODE` to ensure successful execution of the subprogram.

3. Data Persistence Methods

- File-Based Persistence:

Input data is read from files like `BFSFIN` and `BFKARTE`.
Transformed data is written to files like `BFBUOUT` and `BFALL`.

- In-Memory Structures:

Data is temporarily stored and processed in structures like `BEBU`, `IN`, and `GUSSKO`.

This section provides a comprehensive overview of the data structures, file operations, and data flow in `P3BFZ01`, ensuring a clear understanding of the program's functionality and implementation details.

5. Control Flow

Section 5: Main Process Flow and Business Logic

This section provides a comprehensive walkthrough of the main process flow in the program P3BFZ01. It details the step-by-step execution of the program, decision points, loop structures, conditional logic, and business rule implementations. Specific examples from the code, including procedure names, parameters, and implementation details, are included to ensure clarity and completeness.

A. Main Process Flow

The program P3BFZ01 processes financial and material movement data iteratively. The main process flow is structured as follows:

1. Initialization

- The program begins by initializing variables, opening files, and setting up the environment.
- Key initialization steps:

The WKARTE structure is populated by reading control cards from the BFKARTE file using the procedure U_LESEN_BFKARTE.

```
U_LESEN_BFKARTE: PROC REORDER;  
    READ FILE (BFKARTE) INTO (WKARTE);  
END U_LESEN_BFKARTE;
```

The billing period is determined using the procedure U_P9AT003, which calls the subprogram P9AT003 to fetch the billing date and period from the characteristic table database.

```
CALL P9AT003 (ADDR(P9AT003_S));  
IF P9AT003_S.RETURN_CODE ^= '00' THEN  
    PUT SKIP LIST ('No valid billing period found.');
```

SIGNAL ERROR;

```
ELSE  
    ABR_PER = P9AT003_S.AUS.$JAHR || P9AT003_S.AUS.$MONAT;  
END;
```

2. Main Processing Loop

- The program enters a DO WHILE loop that iterates over records in the BFSFIN input file. The loop continues as long as the SATZ_VORH flag is true.
- Key operations within the loop:

Read Input Record:

Each record from the BFSFIN file is read into the BEBU structure.

```
READ FILE (BFSFIN) INTO (BEBU);
```

Transform Data:

The UMSETZEN_IN procedure transforms the BEBU structure into the IN structure, mapping fields and performing calculations.

Example transformation:

```
IN.MENGE = BEBU.MENGE;  
IN.G_PRS_FW = BEBU.HK_PROP;  
IN.E_PRS_FW = ROUND (IN.G_PRS_FW / IN.MENGE, 3);
```

Evaluate Costs:

The BEWERTEN procedure evaluates costs based on the BEBU structure and updates financial data.

```
BEWERTEN: PROC (PARM) REORDER;  
  IF PARM = '1' THEN  
    IN.E_PRS_FW = ROUND (IN.G_PRS_FW / IN.MENGE, 3);  
  END;  
END BEWERTEN;
```

Write Output:

Transformed and processed data is written to output files (BFBUOUT, BFALL, etc.) using procedures like WRITE_BFBUOUT and WRITE_ISTKOSIN.

3. Finalization

- After processing all records, the program performs cleanup operations:

Writes summary statistics using the PGM_STAT procedure.
Closes all open files.

B. Decision Points and Conditions

1. Conditional Logic in **UMSETZEN_IN**

- The UMSETZEN_IN procedure includes a SELECT statement to handle specific cases based on the value of BEBU.UEBGE.
- Example:

```
SELECT (BEBU.UEBGE);  
  WHEN ('AUSSCHUSS') DO;  
    IN.UEBGR = 'SCRAP';  
    IN.MENGE = BEBU.MENGE * (-1);  
  END;  
  WHEN ('50') DO;  
    IN.UEBGR = 'INVOICE';  
  END;  
  OTHERWISE DO;  
    IN.UEBGR = 'UNKNOWN';  
  END;
```

```
END;
```

2. Cost Evaluation in **BEWERTEN**

- The BEWERTEN procedure evaluates costs based on the PARM parameter.
- Example:

```
IF PARM = '1' THEN
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW / IN.MENGE, 3);
ELSE IF PARM = '2' THEN
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW * 1.1, 3);
END;
```

3. Error Handling in **U_P9AT003**

- The U_P9AT003 procedure checks the return code from the subprogram P9AT003 and handles errors.
- Example:

```
IF P9AT003_S.RETURN_CODE ^= '00' THEN
    PUT SKIP LIST ('No valid billing period found.');
```

```
    SIGNAL ERROR;
END;
```

C. Loop Structures and Their Purposes

1. Main Processing Loop

- Purpose: Iterates over records in the BFSFIN file and processes each record.
- Implementation:

```
DO WHILE (SATZ_VORH);
    READ FILE (BFSFIN) INTO (BEBU);
    CALL UMSETZEN_IN;
    CALL BEWERTEN('1');
    CALL WRITE_BFBUEOUT;
END;
```

2. Counter Update Loop in **PGM_STAT**

- Purpose: Updates counters and processes statistics.
- Implementation:

```
DO I = 1 TO 15;
    DANZ(I) = WANZ(I);
    CALL AUS_BFLISTE;
END;
```

D. Complex Conditional Logic

1. Handling Scrap Records in **UMSETZEN_IN**

- Scrap records (BEBU.UEBGEB = 'AUSSCHUSS') require special handling:

The quantity (MENGE) is negated.
The gross price (G_PRS_FW) is adjusted.

```
IF BEBU.UEBGEB = 'AUSSCHUSS' THEN
    IN.MENGE = BEBU.MENGE * (-1);
    IN.G_PRS_FW = BEBU.HK_PROP * (-1);
END;
```

2. Account Adjustments in **WRITE_BFBUOUT**

- For scrap records, account fields (KT_KOAKTO, KT_HKTO) are adjusted if they start with '4'.

```
IF AT.KT_UEBGEB = 'AUSSCHUSS' THEN
    IF SUBSTR(AT.KT_KOAKTO, 1, 1) = '4' THEN
        AT.KT_KOAKTO = '4000';
    END;
END;
```

E. Business Rule Implementations

1. Billing Period Determination

- Business Rule: The billing period must be fetched from the characteristic table database using the subprogram P9AT003.
- Implementation:

```
CALL P9AT003 (ADDR(P9AT003_S));
IF P9AT003_S.RETURN_CODE ^= '00' THEN
    SIGNAL ERROR;
END;
```

2. Scrap Handling

- Business Rule: Scrap records must have negative quantities and adjusted prices.
- Implementation:

```
IF BEBU.UEBGEB = 'AUSSCHUSS' THEN
    IN.MENGE = BEBU.MENGE * (-1);
    IN.G_PRS_FW = BEBU.HK_PROP * (-1);
END;
```

3. Account Adjustments

- Business Rule: For scrap records, account numbers starting with '4' must be replaced with '4000'.
- Implementation:

```

IF AT.KT_UEBGEB = 'AUSSCHUSS' THEN
    IF SUBSTR(AT.KT_KOAKTO, 1, 1) = '4' THEN
        AT.KT_KOAKTO = '4000';
    END;
END;

```

4. Cost Evaluation

- Business Rule: Costs must be evaluated based on predefined modes (PARM values).
- Implementation:

```

IF PARM = '1' THEN
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW / IN.MENGE, 3);
ELSE IF PARM = '2' THEN
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW * 1.1, 3);
END;

```

This section provides a detailed breakdown of the main process flow, decision points, loop structures, conditional logic, and business rules in P3BFZ01. By including specific examples and implementation details, it ensures a clear understanding of the program's functionality.

6. Error Handling

Section 6: Error Handling, Recovery, Logging, and User Notifications

This section provides a detailed explanation of the error handling mechanisms, recovery procedures, logging implementations, and user notification strategies used in the program P3BFZ01. Specific examples from the code, including procedure names, parameters, and business rules, are documented to ensure clarity and completeness.

A. Error Types and Handling

1. File Operation Errors

- Error Type: Issues during file read or write operations.
- Example: In the procedure U_LESEN_BFKARTE, the program reads data from the file BFKARTE into the structure WKARTE. If the file is unavailable or corrupted, the program may encounter a runtime error.

```

READ FILE (BFKARTE) INTO (WKARTE);

```

- Handling:

The program does not explicitly handle file read errors in this procedure. However, the absence of error handling could lead to program termination if the file is inaccessible.

Improvement Suggestion: Implement an ON ERROR block to handle file-related errors gracefully.

2. Database Connection Errors

- Error Type: Failure to connect to the database.
- Example: In the procedure U_CONNECT, the program connects to a database server using the parameter V_SERVER. If the connection fails, the program does not explicitly handle the error.

```
U_CONNECT: PROC (V_SERVER) REORDER;
```

- Handling:

The procedure does not include error handling for connection failures. A return code could be used to indicate the success or failure of the connection.

Improvement Suggestion: Add error logging and user notification mechanisms to inform the user of the connection failure.

3. Subprogram Return Code Errors

- Error Type: Subprograms returning non-zero or unexpected return codes.
- Example: In the procedure U_P9AT003, the subprogram P9AT003 is called to determine the billing period. If the return code is not '00', an error is logged, and the program signals an error.

```
CALL P9AT003 (ADDR(P9AT003_S));  
IF P9AT003_S.RETURN_CODE ^= '00' THEN  
    PUT SKIP LIST ('No valid billing period found.');
```

```
    SIGNAL ERROR;
```

```
END;
```

- Handling:

The program logs the error message: "No valid billing period found."

The SIGNAL ERROR statement terminates the program.

- Business Rule: If no valid billing period is found, the program must halt further processing to prevent incorrect data handling.

4. Division by Zero

- Error Type: Division by zero during calculations.
- Example: In the procedure ERMITTELN_IN_E_PRS, the program calculates E_PRS_FW and E_PRS_ABW by dividing G_PRS_FW and G_PRS_ABW by MENGE. If MENGE is zero, the program assigns default values instead of performing the division.

```
IF IN.MENGE = 0 THEN DO;  
    IN.E_PRS_FW = IN.G_PRS_FW;  
    IN.E_PRS_ABW = IN.G_PRS_ABW;  
END;  
ELSE DO;  
    IN.E_PRS_FW = ROUND(IN.G_PRS_FW / IN.MENGE, 3);  
    IN.E_PRS_ABW = ROUND(IN.G_PRS_ABW / IN.MENGE, 3);  
END;
```

- Handling:

The program explicitly checks if MENGE is zero and assigns default values to avoid a division by zero error.

B. Recovery Procedures

1. Database Connection Recovery

- Procedure: U_CONNECT
- Recovery Steps:

Retry the connection with a delay if the initial attempt fails.
Log the failure and notify the user with the server name and error code.
Example Recovery Logic:

```
ON ERROR BEGIN;  
    PUT SKIP LIST ('Database connection failed for server: ', V  
    SIGNAL ERROR;  
END;
```

2. File Operation Recovery

- Procedure: U_LESEN_BFKARTE
- Recovery Steps:

Check if the file exists before attempting to read.
If the file is missing, log the error and notify the user.
Example Recovery Logic:

```
ON ERROR BEGIN;  
    PUT SKIP LIST ('Error reading file BFKARTE. File may be mis  
    SIGNAL ERROR;  
END;
```

3. Subprogram Error Recovery

- Procedure: U_P9AT003
- Recovery Steps:

If the return code from P9AT003 is not '00', log the error and terminate the program to prevent further processing.

C. Logging Mechanisms

1. Error Logging

- Implementation:

The program uses PUT SKIP LIST statements to log error messages to the console or a log file.
Example:


```
PUT SKIP LIST ('No valid billing period found.');
```

2. Debugging Logs

- Procedure: ERMITTELN_IN_E_PRS
- Implementation:

Logs the calculated values of E_PRS_FW and E_PRS_ABW for debugging purposes.

Example:

```
PUT SKIP LIST ('E_PRS_FW: ', IN.E_PRS_FW, ' E_PRS_ABW: ', IN.E_
```

3. File Write Logs

- Procedure: WRITE_BFBUOUT
- Implementation:

Logs the successful write operation to the BFBUOUT file.

Example:

```
PUT SKIP LIST ('Record written to BFBUOUT: ', AT.KT_KOAKTO);
```

D. User Notifications

1. Error Notifications

- Procedure: U_P9AT003
- Implementation:

Notifies the user when no valid billing period is found.

Example:

```
PUT SKIP LIST ('No valid billing period found. Please check the
```

2. Connection Failure Notifications

- Procedure: U_CONNECT
- Implementation:

Notifies the user of a database connection failure with the server name.

Example:

```
PUT SKIP LIST ('Database connection failed for server: ', V_SEF
```

3. File Operation Notifications

- Procedure: U_LESEN_BFKARTE
- Implementation:

Notifies the user if the file BFKARTE is missing or corrupted.

Example:

E. Business Rules for Error Handling

1. Billing Period Validation

- Rule: If no valid billing period is found, terminate the program to prevent incorrect data processing.
- Implementation:

```
IF P9AT003_S.RETURN_CODE ^= '00' THEN
    PUT SKIP LIST ('No valid billing period found.');
```

```
    SIGNAL ERROR;
END;
```

2. Scrap Record Handling

- Rule: For scrap records, adjust account fields starting with '4' to '4000'.
- Implementation:

```
IF AT.KT_UEBGEB = 'AUSSCHUSS' THEN
    IF SUBSTR(AT.KT_KOAKTO, 1, 1) = '4' THEN
        AT.KT_KOAKTO = '4000';
    END;
```

```
END;
```

3. Division by Zero Prevention

- Rule: Avoid division by zero by assigning default values when the denominator is zero.
- Implementation:

```
IF IN.MENGE = 0 THEN DO;
    IN.E_PRS_FW = IN.G_PRS_FW;
    IN.E_PRS_ABW = IN.G_PRS_ABW;
```

```
END;
```

This section documents the error handling, recovery procedures, logging mechanisms, and user notifications in P3BFZ01. By providing specific examples and implementation details, it ensures a clear understanding of how the program manages errors and communicates with users.

Section 7: External System Dependencies, Data Exchange Formats, and Communication Protocols

This section provides a comprehensive overview of the external system dependencies, data exchange formats, and communication protocols used in the program P3BFZ01. Specific

examples from the code, including procedure names, parameters, and business rules, are documented to ensure clarity and completeness.

A. External System Dependencies

The program P3BFZ01 interacts with several external systems and relies on specific dependencies to perform its operations. These dependencies include database connections, external subprograms, and file systems.

1. Database Dependencies

- DB2 Database:

The program connects to a DB2 database to retrieve and process financial and material movement data.

Procedure: U_CONNECT

Parameters:

V_SERVER: Specifies the server name for the DB2 connection.

Implementation:

```
U_CONNECT: PROC (V_SERVER) REORDER;
```

Business Rule:

The program must establish a one-time connection to the DB2 system using the provided server name.

Example:

In version V13, the connection to the server 'BMWAG_P_DB2P1' was removed, indicating a change in the database dependency.

2. External Subprograms

- The program relies on several external subprograms for specific operations:

P9AT003:

Determines the current billing period from the characteristic table database.

Procedure: U_P9AT003

Parameters:

P9AT003_S: A structure containing input and output fields for the subprogram.

Example:

```
CALL P9AT003 (ADDR(P9AT003_S));
```

Business Rule:

If no valid billing period is found, the program must terminate further processing.

P9BF150:

Evaluates costs and updates financial data.

Procedure: U_P9BF150

Parameters:

PARM: Specifies the evaluation mode.

Example:

```
U_P9BF150: PROC (PARM) REORDER;
```

3. File System Dependencies

- The program interacts with several input and output files:

Input Files:

BFKARTE: Contains control cards for processing.

Procedure: U_LESEN_BFKARTE

Implementation:

```
READ FILE (BFKARTE) INTO (WKARTE);
```

Output Files:

BFBUOUT: Stores financial booking records.

Procedure: WRITE_BFBUOUT

Implementation:

```
WRITE FILE (BFBUOUT) FROM (AT);
```

B. Data Exchange Formats

The program exchanges data using predefined structures and formats. These formats ensure consistency and compatibility with external systems and subprograms.

1. Structure Definitions

- **WKARTE:**

Used to store control card data read from the BFKARTE file.

Fields:

PROGRAMM: Program name.

TEST: Test flag.

Example:

```
READ FILE (BFKARTE) INTO (WKARTE);
```

- **P9AT003_S:**

Used for communication with the P9AT003 subprogram.

Fields:

RETURN_CODE: Indicates success or failure.

AUS.\$JAHR, AUS.\$MONAT, AUS.\$TAG: Billing date components

Example:

```
P9AT003_S.RETURN_CODE = '';  
CALL P9AT003 (ADDR(P9AT003_S));
```

- **BEBU:**

Represents financial data.

Fields:

MANDANT: Client identifier.

MENGE: Quantity.

KTO: Account number.

Example:

```
IN.MENGE = BEBU.MENGE;
```

- **AT:**

Represents transformed financial data for output.

Fields:

KT_KOAKTO: Account number for output.

KT_HKTO: Secondary account number.

Example:

```
WRITE FILE (BFBUOUT) FROM (AT);
```

2. File Formats

- Input File: **BFKARTE:**

Fixed block size of 80.

Contains control card data in a structured format.

- Output File: **BFBUOUT:**

Fixed block size of 300.

Stores financial booking records in a structured format.

C. Communication Protocols

The program uses specific communication protocols to interact with external systems and subprograms. These protocols ensure reliable data exchange and error handling.

1. Database Communication

- Protocol: SQL-based communication with the DB2 database.
- Implementation:

The program includes the SQLCA structure for SQL communication.
Example:

```
EXEC SQL INCLUDE SQLCA;
```

2. Subprogram Communication

- Protocol: Parameter-based communication using predefined structures.
- Example:

The P9AT003 subprogram is called with the address of the P9AT003_S structure.
Implementation:

```
CALL P9AT003 (ADDR(P9AT003_S));
```

3. File Communication

- Protocol: Sequential file read and write operations.
- Example:

Reading control cards from the BFKARTE file:

```
READ FILE (BFKARTE) INTO (WKARTE);
```

Writing booking records to the BFBUOUT file:

```
WRITE FILE (BFBUOUT) FROM (AT);
```

D. Business Rules for Data Exchange

1. Billing Period Validation

- Rule: If no valid billing period is found, terminate the program.
- Implementation:

```
IF P9AT003_S.RETURN_CODE ^= '00' THEN  
    SIGNAL ERROR;
```

2. Scrap Record Handling

- Rule: For scrap records, adjust account fields starting with '4' to '4000'.
- Implementation:

```
IF AT.KT_UEBGEB = 'AUSSCHUSS' THEN  
    IF SUBSTR(AT.KT_KOAKTO, 1, 1) = '4' THEN  
        AT.KT_KOAKTO = '4000';  
    END;
```

3. Division by Zero Prevention

- Rule: Avoid division by zero by assigning default values when the denominator is zero.
- Implementation:

```
IF IN.MENGE = 0 THEN DO;  
    IN.E_PRS_FW = IN.G_PRS_FW;  
    IN.E_PRS_ABW = IN.G_PRS_ABW;  
END;
```

This section documents the external system dependencies, data exchange formats, and communication protocols in P3BFZ01. By providing specific examples and implementation details, it ensures a clear understanding of how the program interacts with external systems and processes data.