VSPKF31-cbl-0.4.4

Report date: *2025-01-27T17:13:15.897071647Z*

User ID: *a458f4f8-3061-70b4-d13b-eed72760d7b0*

AWS Account ID: *156041441283*

Table of Contents

Documentation

1. High Level Overview Of This Program

1.1. Program Name

VSPKF31

1.2. Main Purpose

The main purpose of VSPKF31 is to serve as an interface program to import order data from IVSR (Integrated Vehicle Sales and Registration) for loading and updating the vehicle database. It processes incoming XML data, updates various segments of the vehicle database, and handles different aspects of vehicle orders such as registration, dealer information, and status changes.

1.3. Business Context

This program operates in the context of vehicle sales and registration management. It is a crucial component in the process of updating vehicle information in the database based on data received from the IVSR system. The program handles various aspects of vehicle orders, including status updates, dealer changes, and registration information, which are essential for tracking and managing vehicle sales and inventory across different markets and subsidiaries.

1.4. Key Features

1. XML Parsing: The program parses incoming XML data (MQS-BUFFER) to extract vehicle order information.
2. Database Updates: It updates multiple segments of the vehicle database (VCZ, VCC, VCP, VCU, VC3, VCG, VCW, VCR, VCB, VCN, VCY) based on the parsed data.
3. Status Management: The program handles various vehicle statuses, including current status, sold stock status, and tourist status (VCZ-STATUS-CURRENT, VCZ-STATUS-SOLD-STOCK, VCZ-STATUS-TOURIST).
4. Dealer Information Processing: It manages dealer-related information, including transfers and updates to dealer codes (VCZ-CURRENT-DEALER-CODE, VCY-DEALER-TRANSFER-FROM, VCY-DEALER-TRANSFER-TO).
5. Registration Handling: The program processes vehicle registration information, including dates and registration numbers (VCZ-DATE-REGISTRATION, VCZ-REGISTRATION-NO).
6. Market-Specific Logic: It includes specific logic for different markets (e.g., Japan, Italy, Belgium) to handle unique requirements.
7. Invoice and Credit Processing: The program manages invoice and credit information related to vehicle orders (VCZ-SUBS-INVOICE-NO, VCZ-DATE-SUBS-INVOICE).

8. Error Handling and Logging: It includes error handling mechanisms and logging functionality to track issues during processing.

1.5. Input Output

Input Tables:
1. OIT.TOIO01_ORA: This table is used to retrieve order information for vehicles. It contains data such as vehicle state, product type, order status, and processing type, which are essential for updating the vehicle database.

Output Tables:
N/A

Input Files:
1. VSVBMWI: This file is referenced in the program but its specific use is not clear from the provided code.

Output Files:
1. VSVKF31: This file is used to write error information when processing vehicle orders.
2. VSVKJWS: This file is used to write updates related to the vehicle database, particularly for orders with status greater than 153.

Miscellaneous Input:
1. MQS Queue (MQS-IVSR-QUEUE): The program receives XML data through an MQS (Message Queuing System) queue, which contains the vehicle order information to be processed.
2. TS Queue (TRGTRACE): This queue is used to read trace information for debugging purposes.

Miscellaneous Output:
1. TS Queue (PODLOG): This queue is used to write log information during the processing of vehicle orders.
2. TS Queue (TRANSF): This queue is used to write transfer information when a vehicle order is transferred between dealers.
3. Email Alerts: The program sends email alerts in certain error conditions or for specific processing scenarios.

1.6. Integration

The program does not directly interact with any external systems beyond the databases and files it accesses. It primarily operates within the vehicle sales and registration system, processing data received through the MQS queue and updating the internal vehicle database.

2. Program Logic And Functionality

2.1. Program Logic1

This rule governs the assignment and updating of vehicle data in the system. It processes input data to populate or modify three main data segments: the primary vehicle information, color and upholstery details, and model-specific data including options.

For the primary vehicle information:
- The rule updates or inserts various fields such as the current status, status date, production order number, and vehicle identification numbers based on the input data.
- It performs calculations and assignments, such as setting the status date if a valid value is provided.
- The rule makes decisions based on conditions, like determining the registration status based on the vehicle's state and market.

For the color and upholstery details:
- The rule assigns values to the color code and upholstery code from the corresponding fields in the primary vehicle information.

For the model-specific data:
- The rule populates fields like the model code, color code, upholstery code, and various option-related fields based on the input data and calculations.
- It makes decisions on how to set flags, such as determining if a color is an individual option, based on the input data.

These assignment rules ensure that the vehicle data segments are accurately maintained with the most up-to-date information from the input, which is crucial for maintaining precise vehicle records in the database.

2.2. Program Logic2

This rule manages the handling and updating of vehicle options data. The logic involves the following steps:

1. The rule first determines if the current operation is modifying an existing order or inserting a new one.

2. It then processes the option-related input data, which includes details such as option codes, equipment flags, package flags, and usage types.

3. For each valid option, the rule populates a data structure with the corresponding information.

4. If the operation is modifying an existing order, the rule first removes the old options data before inserting the new ones.

5. The rule then inserts the new options data into the data structure.

6. This process ensures that the vehicle options are accurately updated or added to the vehicle record.

The rule is crucial for maintaining the correct configuration of vehicle options for each order, which impacts various aspects of the vehicle production and sales process. The data structure serves as a repository for this options data, allowing the system to track and manage the specific features and configurations for each vehicle order.

2.3. Program Logic3

This rule is designed to maintain a historical record of vehicle status changes. Whenever a vehicle's status is updated, the rule triggers a process to capture and store this change. The rule ensures that each status modification is recorded with the new status and the date of the change. This creates a comprehensive timeline of a vehicle's journey through various stages in the system.

The rule operates by first retrieving the current vehicle record. It then prepares a new status record with the updated information. Finally, it inserts this new record into the status history. This process allows for the tracking of every status transition a vehicle undergoes.

By implementing this rule, the system can provide valuable insights into the lifecycle of each vehicle. This historical data can be crucial for various business purposes such as:

1. Analyzing the typical progression of vehicles through different statuses
2. Identifying potential bottlenecks in the vehicle processing pipeline
3. Providing accurate status information to customers or internal stakeholders
4. Supporting audit requirements by maintaining a detailed status change log
5. Enabling data-driven decision making based on vehicle status patterns

The rule plays a vital role in maintaining data integrity and providing a reliable source of information for vehicle status history. This can be particularly useful for customer service, operational efficiency, and strategic planning within the organization.

2.4. Program Logic4

This rule governs the assignment of customer information for vehicles based on specific market conditions and vehicle statuses. For the Austrian market, it combines the customer's full name and first name into a single field, using a special delimiter. In other markets, except Japan, the rule handles customer names differently depending on the country and vehicle status. For instance, in Belgium, when the vehicle status meets certain criteria, it separates the customer's first name and surname into distinct fields. For cars and motorcycles with a particular status, it assigns the customer's first name and surname to separate fields, with slight variations between vehicle types. The Japanese market has a unique treatment, where existing customer name information is preserved without modification. Additionally, the rule copies postal code and address details from previous records to new ones, ensuring continuity of customer location data. As a final step, it validates and corrects the customer number field, setting it to zero if it's not a valid numeric value. This rule ensures that customer data is formatted and stored consistently across different markets and vehicle types, while accommodating specific regional requirements.

2.5. Program Logic5

This rule manages the processing of dealer-related information separately from the main vehicle data. It assigns specific values to various dealer data fields in a dedicated structure. The rule sets the project and subsidiary identifiers, initializes the address type, and populates the dealer's name, address components (including multiple address lines), postal code, and a unique domestic dealer number. By doing so, the rule ensures that comprehensive and accurate dealer information is associated with each vehicle record. This structured approach to handling dealer data maintains consistency and integrity within the system, allowing for efficient retrieval and management of dealer-specific details in relation to vehicle records.

2.6. Program Logic6

This rule manages the process of dealer transfers for vehicles. When a vehicle is transferred from one dealer to another, the rule records key information about the transfer. It captures the date of the transfer, the dealer the vehicle is being transferred from, and the dealer it's being transferred to. The rule also tracks whether this is the first transfer for the vehicle or a subsequent one.

The rule is triggered when there's a change in the current dealer associated with a vehicle. It then creates a new transfer record with the relevant details. This includes assigning a transfer number, recording the current date as the transfer date, and storing the dealer codes for both the originating and receiving dealers.

Additionally, the rule updates the assignment status of the vehicle and sets a transfer indicator. If this is the first time the vehicle has been transferred, it marks this accordingly. This information helps in maintaining a comprehensive history of a vehicle's movement through the dealer network.

By implementing this rule, the system can effectively track vehicle movements between dealers, which is crucial for inventory management, dealer performance analysis, and maintaining accurate vehicle histories.

2.7. Program Logic7

This rule governs the assignment of retail and wholesale dealer information based on various conditions. For markets outside Italy, it sets the retail date if available and assigns the current dealer to both retail and wholesale roles. In Italy, the rule has special handling for vehicles with a specific status, setting retail dates and dealer codes accordingly. It also manages cases where the retail dealer code needs to be cleared.

The rule further processes wholesale invoice information. When a valid invoice status is present, it updates the wholesale dealer code and pricing date using invoice details. For certain invoice statuses, it clears both the wholesale dealer code and pricing date.

This logic ensures that dealer information is accurately maintained and updated based on market-specific requirements, vehicle status, and invoice conditions, contributing to proper tracking and management of dealer-related data in the vehicle sales process.

2.8. Program Logic8

This rule manages the processing of vehicles in specific states, particularly demo and own use vehicles, as well as handling associated text information. The rule triggers different actions based on the vehicle's state:

1. For demo vehicles (state 'D'):
- The rule updates or inserts demo-related data, including the start and end dates of the demo period, and mileage information.
- If it's a new demo vehicle, a new record is created; otherwise, existing information is updated.

2. For own use vehicles (state 'O'):
- The rule processes data related to vehicles used by the company itself.
- It manages information such as the start and end dates of the own use period.
- The rule also handles special cases for different markets, applying specific logic for countries like Belgium or the Netherlands.

3. For vehicle text information:
- The rule manages additional textual data associated with vehicles.
- It allows for inserting, updating, or deleting text information based on certain conditions.
- This functionality is particularly relevant for markets like Austria, Belgium, and Spain.

The rule ensures that the appropriate data is recorded and maintained for each vehicle state, automating the process of updating vehicle information as its status changes. This helps in maintaining accurate records of vehicle usage, whether for demonstration purposes or company use, and allows for the inclusion of relevant textual information when necessary.

2.9. Program Logic9

This rule is specifically designed to handle transactions for the Italian market. When a transaction is identified as being for Italy, the rule triggers the initialization and population of a special data structure. This structure contains fields and flags that are unique to Italian vehicle sales or registrations.

The rule ensures that the system applies market-specific requirements for Italy, which may differ from other markets. These requirements could include:

1. Specific registration processes for vehicles in Italy
2. Unique tax calculations applicable to Italian vehicle sales
3. Regulatory compliance checks that are specific to the Italian market

By implementing this rule, the system can adapt its behavior to meet the particular needs of the Italian market, ensuring that all transactions comply with local regulations and business practices. This approach allows for flexibility in handling different market requirements within a single system, while maintaining consistency in overall processing.

2.10. Program Logic10

This rule implements a comprehensive logging and tracking system for vehicle order management. It triggers automated processes to record important information, errors, and changes related to vehicle orders. The rule consists of three main components:

1. Order Journaling: The system writes data to a journaling file to maintain a record of significant events and changes in vehicle orders. This ensures a complete audit trail of order modifications and status updates.

2. Error Logging: The rule includes a mechanism to log errors and other critical events related to vehicle orders. This helps in identifying and troubleshooting issues in the order processing system.

3. Dealer Transfer Tracking: The system records information about dealer transfers for vehicles. This component tracks changes in vehicle ownership or location within the dealer network.

These logging and tracking processes support compliance requirements, facilitate auditing, and potentially enable alerting functions within the vehicle order management system. By maintaining detailed records of order changes, errors, and dealer transfers, the rule ensures transparency and accountability in the vehicle ordering process.

2.11. Program Logic11

This rule manages the assignment and storage of payment terms for vehicles. When executed, it first attempts to retrieve existing payment information for a specific vehicle using its unique identifiers. If a record is found, the rule updates it with the new payment period. If no record exists, a new entry is created with the vehicle's details and the specified payment terms. This ensures that each vehicle in the system has associated payment information, which is crucial for financial processing and customer agreements. The rule allows for flexibility in setting different payment terms for different vehicles, potentially based on factors such as vehicle type, customer status, or market conditions. By maintaining this information in a structured format, the system can easily access and manage payment terms for each vehicle, supporting various financial and operational processes within the business.

2.12. Program Logic12

This rule handles the processing of cosmic invoices, which are treated as a special type of invoice in the system. The rule involves several key steps:

1. The rule checks if a cosmic invoice already exists for a given production order.
2. If a cosmic invoice exists, the rule updates the existing invoice data.
3. If no cosmic invoice exists, the rule creates a new cosmic invoice record.
4. The rule assigns various pieces of information to the cosmic invoice, such as invoice numbers, dates, and amounts.
5. It performs validation checks on the cosmic invoice data to ensure accuracy and completeness.
6. The rule may calculate derived values based on the cosmic invoice data, such as totals or other financial metrics.
7. It updates the status of the cosmic invoice based on certain conditions, such as whether it's a new invoice or a credit note.
8. The rule ensures that duplicate cosmic invoices are not created for the same production order.
9. It may trigger additional processes or updates in related systems based on the cosmic invoice data.
10. The rule likely interfaces with other parts of the system to maintain consistency in invoice-related data across different modules.

This specialized handling of cosmic invoices suggests that they have unique business requirements or processes within the organization's operations, necessitating separate logic from standard invoice processing.

3. The Data Flow And Data Dependencies

3.1. Data Flow

- Input data flow:
- MQS message received via VSPJZMQG call and stored in MQS-BUFFER
- MQS message parsed into XML elements and attributes using XML PARSE
- Parsed data stored in various working storage variables (e.g. WS-CUSTOMER-NAME, WS-REGISTRATION-NO, etc.)

- Database reads:
- VCZ segment read using IO10-VCZ-GU
- VCP segment read using IO28-VCP-GHNP
- VCU segment read using IO31-VCU-GHNP
- VC3 segment read using IO20-VC3-GHNP
- VCG segments read using IO15-VCG-GHNP
- VCW segment read using IO42-VCW-GHU
- VCN segment read using IO37-VCN-GU
- VCB segment read using IO22-VCB-GU
- VCR segment read using IO45-VCR-GHU
- MDZ segment read using IO47-MDZ-GHU

- Database writes:
- VCZ segment updated using IO13-VCZ-REPL or inserted using IO12-VCZ-ISRT
- VCC segment inserted using IO21-VCC-ISRT
- VCP segment updated using IO30-VCP-REPL or inserted using IO29-VCP-ISRT
- VCU segment updated using IO32-VCU-REPL or inserted using IO33-VCU-ISRT
- VC3 segment updated using IO19-VC3-REPL or inserted using IO18-VC3-ISRT
- VCG segments deleted using IO17-VCG-DLET and inserted using IO16-VCG-ISRT
- VCW segment updated using IO43-VCW-REPL or inserted using IO41-VCW-ISRT
- VCY segment inserted using IO38-VCY-ISRT
- VCN segment updated using IO35-VCN-REPL or inserted using IO34-VCN-ISRT
- VCB segment updated using IO25-VCB-REPL or inserted using IO24-VCB-ISRT
- VCR segment updated using IO46-VCR-REPL or inserted using IO44-VCR-ISRT
- MDZ segment updated using IO48-MDZ-REPL

- Output data flow:
- Error messages written to temporary storage queue PODLOG
- Email alerts sent via VSPJZSM2 program call
- JWS updates written to VSVKJWS file

3.2. Data Dependency

- The VCZ segment is the main segment updated and contains key vehicle order data. Most other segments depend on VCZ data:

- VCC segment stores status history and depends on VCZ status data
- VCP segment stores product data and depends on VCZ model codes
- VCU segment stores customer data and depends on VCZ customer fields
- VC3 segment stores option data and depends on VCZ model code
- VCG segments store option data and depend on VCZ order number
- VCW segment stores demo/own use data and depends on VCZ vehicle state
- VCY segment stores dealer transfer data and depends on VCZ dealer codes
- VCN segment stores retail data and depends on VCZ status and dealer codes
- VCB segment stores dealer address data and depends on VCZ dealer code
- VCR segment stores demo data and depends on VCZ demo status
- MDZ segment stores model hold data and depends on VCZ model code

- The status in VCZ (VCZ-STATUS-CURRENT) determines which other segments are updated:
- Status >= 153 triggers JWS updates
- Status = 295 or 298 triggers VCN updates
- Status changes trigger VCC inserts

- Vehicle state in VCZ (WS-VEHICLE-STATE) determines VCW and VCR updates

- Dealer codes in VCZ determine VCY, VCN and VCB updates

- Option data parsed from XML determines VCG and VC3 updates

- Customer data parsed from XML determines VCU updates

- The MQS input message drives all updates - fields parsed from XML are used to update the various database segments

4. Database Information

4.1. Database Overview

The program accesses and manipulates the following database table:

Input Table:
1. OIT.TOIO01_ORA: This table is used to retrieve order information based on the vehicle identification number (VIN) and order status.

Output Tables:
N/A

4.2. Database Connectivity

The program uses embedded SQL to interact with the database. The connection and query execution are handled through the following methods:

1. Connection:
- The program uses a dynamic SQL connection method.
- The connection is established using the EXEC SQL CONNECT statement.
- The target database server is determined based on the current environment (production or integration).
- For production (Japan), it connects to the AG-PROD server.
- For integration, it connects to the AG-VSYS server.

2. Query Execution:
- The program uses a cursor (CURS01) to fetch data from the OIT.TOIO01_ORA table.
- The cursor is declared using EXEC SQL DECLARE CURS01 CURSOR FOR statement.
- The cursor is opened using EXEC SQL OPEN CURS01 statement.
- Data is fetched using EXEC SQL FETCH CURS01 INTO statement.
- The cursor is closed using EXEC SQL CLOSE CURS01 statement.

3. Error Handling:
- The program checks the SQLCODE after each SQL operation to determine if it was successful.
- It uses EVALUATE statements to handle different SQLCODE values and set appropriate return codes.

4. Configuration:
- The program determines the current SQL environment using the EXEC SQL SET :CURRDB2 = CURRENT SERVER statement.
- Based on the current server, it decides which database to connect to (AG-PROD or AG-VSYS).

No specific middleware or drivers are explicitly mentioned in the code. The program appears to rely on the standard COBOL SQL precompiler for database interactions.

4.3. Database Schema

The program has the following database schema:

1. OIT.TOIO01_ORA:
- NV_WS_ORDER_ID (VARCHAR): Unique identifier for the order in the wholesale system
- VEHICLE_STATE (CHAR): Current state of the vehicle
- PRODUCT_TYPE (CHAR): Type of product (e.g., car, motorcycle)
- NV_MF_ORDERER_ID (VARCHAR): Identifier for the orderer in the manufacturing system
- ORDERER_DOMESTIC (VARCHAR): Domestic code for the orderer
- IVS_ORDER_STATUS (VARCHAR): Status of the order in the IVS system
- PROCESSING_TYPE (VARCHAR): Type of processing for the order
- MF_ORDER_NO (VARCHAR): Manufacturing order number
- VEHICLE_ID_7 (VARCHAR): Last 7 digits of the Vehicle Identification Number (VIN)
- VEHICLE_ID_10 (VARCHAR): First 10 digits of the Vehicle Identification Number (VIN)

4.4. SQLQueries And Commands

The program VSPKF31 interacts with the database table OIT.TOIO01_ORA using a cursor named CURS01. The SQL statement used to define this cursor is:

```
DECLARE CURS01 CURSOR FOR
SELECT NV_WS_ORDER_ID,
VEHICLE_STATE,
PRODUCT_TYPE,
NV_MF_ORDERER_ID,
ORDERER_DOMESTIC,
IVS_ORDER_STATUS,
PROCESSING_TYPE,
MF_ORDER_NO,
VEHICLE_ID_7,
VEHICLE_ID_10
FROM OIT.TOIO01_ORA
WHERE VEHICLE_ID_7 = :SQL1_VEHICLE_ID_7
and VEHICLE_ID_10 = :SQL1_VEHICLE_ID_10
and IVS_ORDER_STATUS > :SQL1_IVS_ORDER_STATUS
FOR FETCH ONLY
```

Data columns processed:
1. NV_WS_ORDER_ID: Likely represents a unique order identifier
2. VEHICLE_STATE: Indicates the current state of the vehicle
3. PRODUCT_TYPE: Specifies the type of product (vehicle)

4. NV_MF_ORDERER_ID: Possibly represents the manufacturer's orderer ID
5. ORDERER_DOMESTIC: Likely indicates if the orderer is domestic
6. IVS_ORDER_STATUS: Represents the status of the order in the IVS system
7. PROCESSING_TYPE: Indicates the type of processing for the order
8. MF_ORDER_NO: Manufacturer's order number
9. VEHICLE_ID_7: Last 7 digits of the Vehicle Identification Number (VIN)
10. VEHICLE_ID_10: First 10 digits of the VIN

The cursor is used to fetch data based on the VEHICLE_ID_7, VEHICLE_ID_10, and IVS_ORDER_STATUS conditions. The program manipulates this data by storing it in corresponding variables in the SQL-VAR-O01 structure.

4.5. Data Flow

The data flow between the program and the database occurs primarily through the CURS01 cursor. The process involves the following steps:

1. Data Retrieval:
- The program sets the input parameters SQL1_VEHICLE_ID_7, SQL1_VEHICLE_ID_10, and SQL1_IVS_ORDER_STATUS.
- It then opens the cursor CURS01 using the EXEC SQL OPEN CURS01 statement.
- Data is fetched from the OIT.TOIO01_ORA table using the EXEC SQL FETCH CURS01 INTO :SQL-VAR-O01 statement.
- The fetched data is stored in the SQL-VAR-O01 structure, which includes fields corresponding to the selected columns.

2. Data Processing:
- After successful fetch, the program moves the retrieved data to the ZC30 structure for further processing.
- For example, SQL1_NV_WS_ORDER_ID is moved to ZC30-NV-WS-ORDER-ID, SQL1_VEHICLE_STATE to ZC30-VEHICLE-STATE, etc.

3. Error Handling:
- The program checks the SQLCODE after each database operation (OPEN, FETCH, CLOSE) to determine if the operation was successful.

4. Cursor Closing:
- After processing, the cursor is closed using the EXEC SQL CLOSE CURS01 statement.

There are no explicit UPDATE, INSERT, or DELETE operations in the provided code snippet. The interaction appears to be read-only, focusing on retrieving and processing data from the OIT.TOIO01_ORA table.

4.6. Error Handling And Transaction Management

The program implements error handling for database operations, but there's no explicit transaction management (commit/rollback) visible in the provided code. Here's how errors are handled:

1. For cursor operations (OPEN, FETCH, CLOSE):
- After each operation, the program checks the SQLCODE.
- If SQLCODE is zero, it's considered successful, and 'OK' is moved to ZC30-RC.
- For FETCH operations, SQLCODE +100 is handled separately to indicate end of data.
- For any other SQLCODE, it's considered an error:
- 'NOTOK' or 'EOF' is moved to ZC30-RC
- The SQLCODE is stored in ZC30-SQL-CODE
- An error message is displayed, including the operation and the SQLCODE

2. Error Logging:
- When an error occurs, the program displays an alert message with the operation name and the SQLCODE.
- For example: "SQL ALERT OPEN CURSOR01 - SQLCODE IS " followed by the actual SQLCODE.

3. Error Recovery:
- The program sets a flag WS-IVSR-ORDER-FOUND to 'N' when a FETCH operation fails or reaches the end of data.
- This allows the main program logic to handle cases where no data is found or an error occurs during data retrieval.

4. Transaction Management:
- There are no explicit COMMIT or ROLLBACK statements in the provided code.
- The cursor is declared with "FOR FETCH ONLY", indicating read-only operations, which typically don't require explicit transaction management.

While the error handling is present, the lack of explicit transaction management suggests that this part of the program is primarily concerned with reading data, not modifying it. If data modifications occur elsewhere in the program, additional transaction management practices would be necessary.

4.7. Additional Considerations

1. Connection Management:
- The program includes a section (YY00-DB2-CONNECT) for managing database connections.
- It checks the current server (CURRDB2) and connects to different databases based on the environment (production, integration, etc.).
- This allows for flexibility in connecting to different environments without changing the core program logic.

2. Cursor Usage:

- The program uses a cursor (CURS01) for fetching data, which is a best practice for retrieving multiple rows or when the result set size is unknown.
- The cursor is declared as "FOR FETCH ONLY", which can improve performance for read-only operations.

3. Parameter Usage:
- The program uses host variables (prefixed with :) in the SQL statements, which is a good practice for parameterized queries.

4. Error Handling Granularity:
- The program checks SQLCODE after each database operation, providing fine-grained error handling.

5. Data Integrity:
- The WHERE clause in the cursor declaration uses multiple conditions (VEHICLE_ID_7, VEHICLE_ID_10, IVS_ORDER_STATUS), which helps in ensuring data integrity and fetching the correct records.

6. Code Organization:
- Database operations are organized into separate sections (OPEN, FETCH, CLOSE), improving code readability and maintainability.

7. Performance Considerations:
- The use of a cursor with specific WHERE conditions can help in optimizing the database query performance.

8. Logging and Debugging:
- The program includes DISPLAY statements for logging SQL operations and their outcomes, which can be helpful for debugging and monitoring.

9. Scalability:
- The program's structure allows for easy addition of more complex SQL operations or additional cursors if needed in the future.

While these practices contribute to a well-structured database interaction, there's always room for improvement, such as implementing more robust transaction management if data modifications are introduced, or considering the use of prepared statements for frequently executed queries.

5. Error And Abend Handling

5.1. Overall Strategy

The program VSPKF31 employs a comprehensive error handling strategy that combines various techniques:
1. Use of CICS error handling commands (HANDLE CONDITION)
2. DL/I status code checks
3. Custom error sections for different types of errors (e.g., TS queue errors, VSAM errors)
4. Abend handling routines
5. Error logging mechanisms

This multi-layered approach ensures that errors are detected, logged, and handled appropriately throughout the program's execution.

5.2. Common Error Conditions

5.2.1. List Errors

1. CICS errors: INVREQ, IOERR, ITEMERR, LENGERR, NOSPACE, QIDERR
2. DL/I errors: Various status codes (e.g., 'GE', 'GB')
3. VSAM errors: DSIDERR, DUPREC, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTOPEN, DISABLED
4. MQS errors
5. Database access errors (SQL errors)
6. Data validation errors
7. Abend conditions

5.2.2. Error Sources

1. CICS errors: Temporary storage operations, queue operations
2. DL/I errors: Database operations (read, write, update, delete)
3. VSAM errors: File operations
4. MQS errors: Message queue operations
5. Database errors: SQL operations, connection issues
6. Data validation errors: Invalid input data or unexpected data formats
7. Abend conditions: Various program execution errors

5.3. Error Detection Mechanisms

5.3.1. Detect Methods

1. CICS HANDLE CONDITION commands for detecting CICS-specific errors
2. Checking DL/I status codes after database operations
3. Evaluating SQLCODE after SQL operations

4. Checking return codes from called subprograms (e.g., VSPJZMQG for MQS operations)
5. Custom error detection routines (e.g., U460-HANDLE-CONDITION-TS, U470-HANDLE-CONDITION-VSAM)
6. Evaluating specific fields or conditions in the program logic

5.3.2. Functions And Exception Check

1. EXEC CICS HANDLE CONDITION: Used for CICS error handling (e.g., in U460-HANDLE-CONDITION-TS)
2. CALL 'CBLTDLI': Used for DL/I operations, with status code checks afterward
3. EXEC SQL statements: Used for database operations, with SQLCODE checks
4. Custom error sections: X100-INVREQ, X101-IOERR, X102-ITEMERR, etc., for handling specific error conditions
5. Abend handling routines: XX-ABEND-RETURN, X600-ABEND-RETURN for managing program termination
6. Error logging routines: U410-WRITE-LOG-QUEUE for writing error logs

5.4. Error Responses And Recovery Actions

5.4.1. Immediate Response

1. Error logging: The program uses U410-WRITE-LOG-QUEUE to log errors to a temporary storage queue
2. Setting error flags: Various error flags are set (e.g., WS-QID-ITEM-ERROR, WS-TRG-ITEM-ERROR)
3. Populating error information: Error details are stored in WSA-ABEND-CODE, WSA-VSAM-CODE, etc.
4. Calling error handling routines: Specific error handling sections are called based on the error type
5. Sending email alerts: U015-OUTPUT-EMAIL is used to send email notifications for certain errors
6. Displaying error messages: Error messages are prepared and displayed (e.g., using THIS-MSG)

5.4.2. Recovery Procedures

1. Rollback: EXEC CICS SYNCPOINT ROLLBACK is used to rollback database changes in case of errors
2. Retry mechanisms: In some cases, the program attempts to retry operations (e.g., U430-READ-U00-NO)
3. Graceful termination: X600-ABEND-RETURN is used to handle abend conditions and terminate the program
4. Error bypassing: In some cases, the program continues execution after logging the error (e.g., ignoring certain errors in U010-VCZ)
5. Alternative flow: The program may take alternative execution paths based on error conditions (e.g., in U000-UPDATE-DATABASE)

5.5. Error Logging Mechanisms

5.5.1. Mechanisms Details

1. Temporary Storage Queue: The program uses a TS queue named 'PODLOG' to log errors (U410-WRITE-LOG-QUEUE)
2. File logging: The program writes error information to a file named 'VSVKF31' (IO49-WRITE-F31)
3. CICS error logging: CICS built-in error logging is utilized through HANDLE CONDITION commands
4. Database logging: Some error information is stored in database tables (e.g., VCZ, VCB segments)
5. In-memory logging: Error information is stored in working storage variables for further processing

5.5.2. Error Logs Format And Content

1. TS Queue log format:
- LOG-PRODNO: Production order number
- LOG-ELEMENT: Current element being processed
- LOG-ATTRIBUTE: Current attribute being processed
- LOG-TEXT: Error message text
- LOG-TIMESTAMP: Timestamp of the error

2. File log format (F31-REC):
- F31-KEY: Unique identifier
- F31-CHASSIS: Vehicle chassis number
- F31-PRODNO: Production order number
- F31-PRODUCT-TYPE: Product type
- F31-ERROR-MESSAGE: Detailed error message
- F31-ERROR-CODE: Numeric error code
- F31-ERROR-SEVERITY: Error severity level
- F31-TIMESTAMP: Timestamp of the error

3. In-memory log format:
- WSA-ABEND-CODE: Abend code
- WSA-VSAM-CODE: VSAM error code
- WSA-DBD-NAME: Database name
- WSA-SEG-LEVEL: Segment level
- WSA-STATUS-CODE: Status code
- WSA-PROC-OPTIONS: Processing options
- WSA-SEG-NAME-FB: Segment name feedback
- WSA-LENGTH-FB-KEY: Length of feedback key
- WSA-NUMB-SENSEG: Number of sensitive segments
- WSA-KEY-FB: Key feedback area

5.6. User Notifications And Interface Handling

5.6.1. Errors Communication

1. Email notifications: The program uses U015-OUTPUT-EMAIL to send email alerts for critical errors

2. Error messages: The program prepares error messages (THIS-MSG) which can be displayed or logged
3. Status codes: Various status codes are set (e.g., VCZ-STATUS-CURRENT) which can be used by calling programs or interfaces to determine the operation result
4. Return codes: The program sets return codes (e.g., MQS-RETURN-CODE) which can be checked by calling programs
5. Abend codes: For severe errors, the program sets abend codes (WSA-ABEND-CODE) which can trigger system-level error handling

5.6.2. Error Messages Interface

The program doesn't directly interface with end-users, but it prepares error information that can be used by calling programs or interfaces:

1. Error message preparation: The program formats error messages in THIS-MSG, which can be displayed by calling programs
2. Protocol messages: U200-PROTOCOL is used to prepare protocol messages that can be displayed or logged
3. Email alerts: U015-OUTPUT-EMAIL prepares and sends email alerts with error information
4. Status codes: Various status fields (e.g., VCZ-STATUS-CURRENT, VCZ-STATUS-PASSED-TO-IWS) are updated to reflect error conditions
5. Error logs: The program writes detailed error logs that can be accessed by support staff or other systems for analysis and resolution

While the program itself doesn't have a user interface, it provides comprehensive error information that can be utilized by other components of the system to display appropriate error messages or instructions to end-users or system operators.

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 an XML document containing vehicle sales order data. This input is significant as it contains all the necessary information to update or insert vehicle records in the database.

6.1.2. Input Source

The input XML document is received from an MQS (Message Queuing System) queue named 'BMW.CV.IVSR.IVS.XX.POD.MESSAGES', where XX is a country-specific suffix.

6.1.3. Input Format And Structure

6.1.3.1. Field Lengths

The XML document is stored in MQS-BUFFER with a maximum length of 32000 characters. Specific field lengths within the XML structure vary based on the data elements, such as:
- VCZ-PRODUCTION-ORDER-NO: 7 characters
- VCZ-SERIAL-NO-11-17: 7 characters
- VCZ-SERIAL-NO-01-10: 10 characters
- VCZ-STATUS-CURRENT: 3 characters

6.1.3.2. Mandatory Fields

Mandatory fields include:
- VCZ-PRODUCTION-ORDER-NO (Production Order Number)
- VCZ-SERIAL-NO-11-17 (Vehicle Identification Number, last 7 digits)
- VCZ-PRODUCT-TYPE (Product Type)
- VCZ-STATUS-CURRENT (Current Status)

6.1.3.3. Data Types

- VCZ-PRODUCTION-ORDER-NO: Numeric
- VCZ-SERIAL-NO-11-17: Alphanumeric
- VCZ-SERIAL-NO-01-10: Alphanumeric
- VCZ-STATUS-CURRENT: Numeric
- VCZ-DATE-STATUS-CURRENT: Numeric (YYYYMMDD format)
- VCZ-CURRENT-DEALER-CODE: Numeric

6.1.3.4. Additional Requirements

- The input XML must follow a specific structure with elements like <salesOrder>, <orderData>, <vehicleSpecification>, etc.

- Country-specific processing is determined by the VSUB-SUBS-CODE (e.g., ZB for Belgium, ZE for Spain, etc.)

6.2. Output Specifications

6.2.1. Output Purpose And Downstream Usage

The main purpose of the output is to update or insert vehicle records in the IMS database. This includes updating various segments like VCZ (main vehicle data), VCC (status history), VCP (product data), VCU (customer data), VC3 (options), VCG (options), VCW (own use data), VCY (dealer transfer), VCN (retail data), VCB (national data), and VCR (demo data).

6.2.2. Output Destination

The primary output destination is the IMS database, specifically the vehicle database (PCBVC01). Additionally, the program writes to temporary storage queues for logging and tracing purposes, and in some cases, it writes to VSAM files like VSVKJWS for JWS (Job Work Sheet) updates.

6.2.3. Output Format And Structure

6.2.3.1. Formatting Rules

- Database updates follow IMS DL/I call conventions (e.g., GU, GHU, ISRT, REPL, DLET).
- Date fields are generally formatted as YYYYMMDD.
- Numeric fields are zero-padded to their full length.
- Alphanumeric fields are space-padded when necessary.

6.2.3.2. Structure

The output structure varies depending on the database segment being updated:
- VCZ segment: Contains main vehicle data including order number, VIN, status, dates, etc.
- VCC segment: Status history records.
- VCP segment: Product-specific data like color and upholstery codes.
- VCU segment: Customer information.
- VC3 segment: Vehicle options and features.
- VCG segment: Additional options data.
- VCW segment: Own use vehicle data.
- VCY segment: Dealer transfer information.
- VCN segment: Retail data.
- VCB segment: National-specific data.
- VCR segment: Demo vehicle data.

Each segment has its specific layout defined in the respective copybooks (e.g., VSCLEVCZ, VSCLEVCC, etc.).

6.3. Error Handling

6.3.1. Input Data Errors Handling

- XML parsing errors are handled in the XML PARSE EXCEPTION section. The program captures the error details and writes them to a log.

- For critical errors, the program sets an error indicator (POD-ERROR-IND) and prepares an email alert with error details.
- Invalid dealer codes are logged and may trigger email alerts.
- The program uses various error codes (e.g., E002, E009, E015) to identify and handle different types of errors.
- In case of severe errors, the program may perform a rollback (EXEC CICS SYNCPOINT ROLLBACK) to maintain data integrity.

6.3.2. Output Errors Handling

- Database operation errors (e.g., during ISRT, REPL, DLET) are captured and logged.
- The program uses specific error handling routines for different database segments (e.g., XXY-VC-DL1-ERROR, XXY-PM-DL1-ERROR).
- In case of output errors, the program may set specific error codes and perform a rollback.
- For certain error conditions, the program writes error information to temporary storage queues for later analysis.
- Critical errors may result in the program terminating with an abend, using routines like X600-ABEND-RETURN or XX-ABEND-RETURN.

7. Integration Information

7.1. Integration Overview

7.1.1. Purpose Of Integration

The purpose of the MQ integration is to receive vehicle order data from an external system for processing and updating the vehicle database. The CICS integration is used for various system operations, transaction management, and data handling within the program.

7.1.2. Integration Interactions

The program interacts with MQ by performing GET operations to retrieve messages from a queue. It processes the received XML data and updates the vehicle database accordingly. The program interacts with CICS through various commands for temporary storage operations, file handling, and transaction management.

7.2. Integration Type

1. MQ (Message Queue): Data exchange through message queuing
2. CICS (Customer Information Control System): Transaction processing and system services

7.3. Interacted External Systems

1. MQ (Message Queue) system
2. CICS (Customer Information Control System)

7.4. External System Integration

7.4.1. System Type

1. MQ: Message-oriented middleware
2. CICS: Transaction server

7.4.2. Tech Stack

1. MQ: IBM MQ (formerly WebSphere MQ)
2. CICS: IBM CICS Transaction Server

7.4.3. Integration Role Of External System

1. MQ: Provides asynchronous messaging capabilities for receiving vehicle order data
2. CICS: Manages transactions, provides temporary storage, and handles file operations for the program

7.5. External System Data Flow

1. MQ:
- The program retrieves messages from an MQ queue using the GET operation
- The received message contains XML data with vehicle order information
- The program processes the XML data and updates the vehicle database accordingly

2. CICS:
- The program uses CICS commands for temporary storage operations (WRITEQ TS, READQ TS)
- File operations are performed using CICS commands (READ, WRITE, DELETE)
- CICS handles transaction management and provides system services for the program

7.6. External System Interface

1. MQ:
- Interface: MQ API calls through the VSPJZMQG program
- Message format: XML
- Queue name: Stored in MQS-IVSR-QUEUE variable
- GET operation parameters: Defined in GETOPT structure

2. CICS:
- Interface: CICS API commands (EXEC CICS ...)
- Temporary storage operations: WRITEQ TS, READQ TS
- File operations: READ, WRITE, DELETE
- Transaction management: START, RETURN, SYNCPOINT
- Error handling: HANDLE CONDITION

7.7. External System Errors Handling

1. MQ:
- The program checks the MQS-RETURN-CODE after each MQ operation
- If an error occurs, it calls the VSPJZMQE program to handle the error
- The program uses the X100-MQS-ERROR section to process MQ errors

2. CICS:
- The program uses CICS HANDLE CONDITION commands to set up error handling for various CICS operations
- Specific error handling sections are defined for different types of errors (e.g., INVREQ, IOERR, ITEMERR)
- In case of errors, the program may perform actions such as logging the error, rolling back the transaction, or terminating the program
- The U670-LINK-ERR-PGM-AND-ROLLBACK section is used to handle severe errors by linking to an error program and rolling back the transaction

8. Screen Documentation

This program does not have any screen.

9. Programs Called

9.1. Application Programs

N/A

9.2. System Programs

CBLTDLI, CMQTMV, CMQV, SQLCA

9.3. Missing Programs

SUPJO025, VSPJO001, VSPJZMQE, VSPJZMQG, VSPJZSM2, VSPKJ11R, VSPKS02