Vehicle Management System: Creating Actions

Technical Guidelines
Call, Control and Creation of Actions in the Vehicle Management System (VMS)

Release DIMP 4.71 (Discrete Industries & Mill Products)
Automotive 4.0 for VMS 4.0
3.3.1.2 Assigning Vehicle Attributes to Actions ........................................... 41
3.3.2 Processing Customer-Specific Additional Data as Qualifiers .................... 42
3.3.2.1 Defining Qualifier Data ........................................................................ 42
3.3.2.2 Assigning Qualifier Data to Actions .................................................. 42
3.3.3 Qualifier Data Versus Vehicle Attributes ............................................... 43
3.4 Creating and Calling a Prepare Module .................................................. 44
3.4.1 Naming Conventions ............................................................................. 44
3.4.2 Documentation ...................................................................................... 44
3.4.3 Interface ............................................................................................... 45
3.4.4 Source Text ......................................................................................... 47
3.4.4.1 Data Declarations ............................................................................. 47
3.4.4.2 Checks ............................................................................................. 49
3.4.4.2.1 Reading and Checking Preceding Document ................................. 49
3.4.4.2.2 Checks ....................................................................................... 51
3.4.4.3 Filling the Communication Structure .................................................. 52
3.4.4.3.1 Header Data ................................................................................ 52
3.4.4.3.2 Item Data ................................................................................... 53
3.4.4.4 Defining a Template Configuration .................................................... 54
3.4.5 Call ....................................................................................................... 55
3.5 Creating and Calling an Execute Module .................................................. 56
3.5.1 Naming Conventions ............................................................................. 56
3.5.2 Documentation ...................................................................................... 56
3.5.3 Interface ............................................................................................... 57
3.5.4 Source Text ......................................................................................... 59
3.5.4.1 Data Declarations ............................................................................. 59
3.5.4.2 Checks ............................................................................................. 60
3.5.4.2.1 Reading and Checking Preceding Document ................................. 60
3.5.4.2.2 Calling the PREPARE Module ..................................................... 63
3.5.4.3 Creating/Changing/Deleting SAP Standard Documents ....................... 64
3.5.4.4 Updating VMS Document Tables ....................................................... 66
3.5.4.5 Update ............................................................................................. 67
3.5.4.6 Writing Vehicle Data ........................................................................ 68
3.5.4.7 Filling the Communication Structure .................................................. 69
3.5.4.7.1 Header Data ................................................................................ 69
3.5.4.7.2 Item Data ................................................................................... 70
3.5.5 Call ....................................................................................................... 71
3.6 Branching from Vehicle History to Document ............................................ 72

4 Appendix .................................................................................................... 73
4.1 Tables in VMS .......................................................................................... 73
4.1.1 Transaction Data ................................................................................... 73
4.1.2 Master Data .......................................................................................... 74
4.1.3 Customizing Tables ............................................................................. 74
4.1.4 Control Tables ....................................................................................... 76
4.2 Guidelines for Programming in VMS ....................................................... 76
1 Technical Background: Action Processing

1.1 Definition: Action

An action is a business transaction that refers to business processes when you are handling vehicles, for example: create vehicle, change vehicle, create purchase order or create sales order.

A vehicle has two statuses, the primary and the secondary status. The vehicle status changes automatically depending on the action you perform. An action can set the primary and/or the secondary status.

The business processes are defined as action and status sequences in an action matrix that you define in Customizing. An entry in the action matrix defines which actions are allowed for which current vehicle status and which new status is set if the action has been performed successfully. A primary and a secondary matrix is assigned to a vehicle in the Vehicle Management System in order to control the status sequences of primary and secondary statuses.

Actions are defined in Customizing as primary and/or secondary actions. The terms primary and secondary should not be regarded as higher or lower level, in other words, they do not indicate a hierarchy. Whether the actions are primary or secondary depends on the individual business processes of the user. Possible divisions are: purchasing actions versus sales actions or logistical actions (actions that affect the vehicle itself) versus financial actions (actions that affect payment transactions).

If you perform an action in the Vehicle Management System, general checks and functions are executed independently of the chosen action. Examples are: the system checks whether the chosen action is allowed for the current vehicle status, vehicle additional data (attributes, qualifier) is written, a new status is set if the action has been performed successfully and an entry is written in the vehicle history. If you develop your own actions and want to link them to the Vehicle Management System, this knowledge of the processes involved will be helpful to you.

Furthermore, many actions require data to be entered (for example: action CREA) so that they can be performed. In this case the action is started, data is entered and this data is then processed. This interaction requires important check and functions to be executed several times. You must also take this into account when you develop customer-specific actions.

The goal of this documentation is to explain the technical processes involved when actions are called and performed. Moreover, the coding of the relevant function modules and programs will also be elucidated.

Basically-speaking, we differentiate between simple and complex actions.
1.1.1 Simple Action

Simple actions are actions that can only be created by settings that are made in Customizing for the Vehicle Management System. Here, the new action is created and the status sequence of the action matrix integrated (see: 2).

Such actions result in only one status update and one entry in the vehicle history.

The simple actions do not have any action-specific repository objects and therefore do not require any development. A standard screen is automatically used as the action screen, which does not contain any elements apart from the button for executing the action.

1.1.2 Complex Action

A complex action usually consists of a specific action screen, its PBO and PAI modules as well as a PREPARE and an EXECUTE function module. The communication interfaces are fixed but, if necessary, can be enhanced to fulfill customer-specific requirements.

Complex actions are thus actions that contain their own coding or their own subscreens. They allow you to program actions with your own logic (see: 3), which can fulfill a myriad of different tasks.

As a rule, a complex action consists of the following three parts, each of which is action-specific.

- **Action screen**: a separate subscreen (optional) that is displayed on the right-hand side next to the vehicle list on the tab page *Action*. It is nearly always sufficient to use exclusively the PAI and PBO modules delivered with the standard VMS system.

- **EXECUTE function module**: there is usually one function module (VELO<NN>_<ACTION>_EXECUTE) that is responsible for the actual execution of the action. This module, for example, creates an R/3 standard document by calling a BAPI.

  If you only want to write data from the communication structure (see:1.5) or the actions screen to the vehicle table and/or the history table, you do not need your own
EXECUTE module. You only have to make the appropriate settings in Customizing under the IMG activity Define Additional Data for Vehicle (see: 3.3.1).

- **PREPARE function module**: there is the option of using a module (VELO<NN>_<ACTION>_PREPARE) that is called immediately after the action has been called. Amongst other things, this module can make action-specific checks and fill the action screen with default values.

A defined communication structure enables the transfer of data between the actions screen and action processing. You can only use fields of this structure on the action screens. You can enhance the structure to fulfill customer-specific requirements by using the APPEND technique. The structure and enhancement options that this structure offers you is explained in more detail below (see: 1.5).

The graphic displays the process flow for a complex action in transaction VELO:

- If a PREPARE function module exists for the action, this is called first. The action-specific checks are called and data is prepared for the action screen here. The data is copied to the communication structure VLCACTDATA_CS.

- Then an action-specific screen is displayed, if one exists. This screen contains the fields from communication structure VLCACTDATA_CS that are required for the action. The user makes entries in these fields and then chooses the function for executing the action.

- Now the actual action processing is performed, in other words, the EXECUTE function module is called. The action is performed with the data from the communication structure for all vehicles that were copied to internal table VLCDIAVEHI_CT. After several checks, a BAPI that creates an SAP standard document, for example, could be called.

You can define interlinking actions in VMS Customizing (see: 2.7), in other words, actions for which several elementary actions can be performed automatically one after the other. All the relevant PREPARE modules are processed one after the other, then a common action screen is displayed and after the user has entered the data and chosen the function for executing the actions, all the relevant EXECUTE modules are processed.
You can call VMS actions via BAPIs instead of via transaction VELO. The IDoc interface or the web interface, for example, use this technique. If actions are performed via a VMS BAPI, only the EXECUTE modules are used. The checks implemented in the PREPARE modules have to be executed in the EXECUTE modules. In other words:

an action has to be completely described by its EXECUTE module, both logically and as regards the business side.

For reasons of user-friendliness, however, you should not do without important checks in the PREPARE modules. Users in transaction VELO, for example, are informed as soon as possible, i.e. when they call an action (that is, at the call-up point of the PREAPRE module) that an action cannot be performed.

1.1.3 Interlinking Action

An interlinking action is an action that consists of two or more elementary actions that are linked with one another. An interlinking action is defined in VMS Customizing under the IMG activity Define Actions (see: 2.7). These elementary actions can be simple or complex actions.

When the interlinking action is started, all the other elementary actions contained in it are performed in the given order. The system processes each elementary action as a complete action, including setting a new status and updating data.

💡 When you perform an interlinking action, the elementary actions are performed one after the other and posted separately. In other words, each elementary action is handled as a separate LUW (logical unit of work).

The interlinking of actions does not require any coding. However, if each of the individual elementary actions requires different entry fields, you have to create a new action screen that contains all of these fields and assign it to the interlinking action in Customizing under the IMG activity Define Technical Details for Actions.
1.2 VMS Encapsulation Model

This VMS function (package ISAUTO_VLC) has been thematically divided into several function groups. There is a clear-cut distinction between presentation, application and database functionalities. Due to the placing of VMS as an independent and flexible module, all objects have been completely encapsulated.

The five groups marked in red (among others) belong to the action function groups to which the actions have been thematically assigned: VEL010, VEL011, VEL015, VEL016 and VEL017.
VELO15 plays a particularly important role as it is, so to speak, the original action function group. The PAI and the PBO modules which are required for the action screens are defined, for example, in VELO15. All other action function groups simply include these modules (see: 3.1).

VELO09 is also of great importance since general action handling is implemented here.

The VMS R/3 interface and the flow logic required for interaction with the user is stored in function group VEL002. The subroutine USER_COMMAND_4000 is of central importance in this function group (see below); here the function codes are processed, for example the function code for executing an action via which the function module for PREPARE and EXECUTE processing is called.

The function group VEL002 contains the general VMS constant include LVELO02CON.
1.3 Action Processing: Flow

The general technical process flows when calling and performing actions are described below. Following that, the ABAP coding for the individual objects is elucidated and the most important logical steps explained.

- When you call an action in transaction VELO (choose action and then the green check flag, function code ACTN), the subroutine USER_COMMAND_4000 is called in the PAI for screen 4000 (VELO subscreen container, function group VEO02). PREPARE processing is called and the tab page Action is activated here.

- Then the list of vehicles to be processed is drawn together and the action screen defined and called in the PBO for screen 5000 (tab page Action). The PBO modules for the action-specific subscreen are called during this step.

- After any necessary data has been entered and the action has been performed (function code AKTN), the PAI for screen 4000 is called again and subroutine USER_COMMAND_4000 run through; EXECUTE processing is called and the action log is written.
• Finally, the vehicle list is adapted in the PBO callup point for screen 5000 (tab page *Action*) and the (success LEDs - light-emitting diodes or indicators - and data changes) and the action screen is called again. Data (for example, the number of the document created) is displayed on the action screen. Furthermore, the user can call the action log.

### 1.3.1 PREPARE Processing

General and action-specific PREPARE processing has the task of performing checks when an action is called and preparing data for the action screen and further processing.

- When an action is performed via a VMS BAPI, PREPARE processing is not called; this type of processing is therefore only relevant when you are using transaction VELO.
- An action does not have to contain a PREPARE module.

The following graphic shows which modules are called during PREPARE processing. This general sequence is the same for all actions. If an action does not have its own PREPARE module, the call of the appropriate module is simply dropped.
Subroutine USER_COMMAND_4000 calls PREPARE processing, i.e. module Velo09_ACTION_DATA_PREPARE for function code ACTN (call action). This module then calls in turn module Velo09_ACTION_DATA_PREPARE_INT for each action to be performed (this can be more than one for interlinking actions). This then starts the following functions:

- the action-specific PREPARE module, if it exists
- method PREPARE_FURTHER_ACTION for BAdl VLC_PREPARE_ACTION; the customer-specific PREPARE modules are called in this BAdl (see: 3.4.5)
- module Velo09_PREPARE_VEHICLE_DATA. If details (see: 3.3) have been defined in Customizing under the IMG activity Define Additional Data for Vehicle) to the effect that an action should prepare additional data (attribute or qualifier), the additional data defined in Customizing is prepared here.

The following graphic shows the data flow in PREPARE processing:

The PREPARE processing contains information on which action should be performed for which vehicles (VLCDIAVEHI_T). Data is prepared and then read and checked by the database. If an error occurs, processing is terminated immediately with an appropriate error message. The data that has been prepared is copied to communication structure VLCACTDATA_S and displayed on the action screen.

Only fields of communication structure VLCACTDATA_S can be displayed on an action screen.

After any data that is required has been entered and the action has been performed (function code AKTN), the communication structure and the vehicle table is transferred to EXECUTE processing (see: 1.3.2).
1.3.1.1 Form USER_COMMAND_4000

Module USER_COMMAND_4000 is the controlling PAI module in transaction VELO (function group VELO02). Specific function codes are evaluated by the user and further processing is triggered.

These are the tasks of the PAI module with reference to the PREPARE callup point (function code ACTN):

- Check whether action exists/read action texts in the logon language
- Check whether the action is allowed for the selected vehicles
- Set and initialize global data (see: 1.6.1) in VEL009 (set chosen action, initialize communication structure VLCACTDATA_CS, transfer vehicle list)
- Call PREPARE processing: VEL09_ACTION_DATA_PREPARE
- Message informing you whether the action can be performed
- Set the LEDs (light-emitting diodes or indicators) in the vehicle list
- Set parameter BEFORE_AFTER_GV in VEL009: this determines whether processing takes place before or after the action This parameter is evaluated in a general PBO module for the action screens in order to control field attributes (for example, ready for input status; see: 3.2.2.1)
- Set variable CONFIGURE_BUTTON_GV according to the settings in Customizing for actions (see: 2.2); this determines whether a function for configuring data is needed on the action screen This variable is evaluated in a general PBO module for the action screens (see: 3.2.2.1)
- If the indicator for starting the configuration immediately is set in Customizing for the actions (see: 2.2), the configuration is read and displayed here before the action screen is processed

The configuration can be displayed in CU50 or in the IPC interface. The IPC interface is normally used; if you set the user parameter VEL0_CU50_ACTIVE = X, the CU50 interface is used.

1.3.1.2 Function Module VEL09_ACTION_DATA_PREPARE

This function module performs some general checks and calls the action-specific PREPARE processing. The individual steps are:

- Read global data (see: 1.6.1) from VEL009 (communication structure VLCACTDATA_CS and list of list of vehicles to be processed)
- Check whether action exists/read action texts in the logon language
- Fill internal table ACTION_LT with the actions to be performed (this can be more than one for an interlinking action)
- Check whether the user is allowed to perform the action(s)
- Function module VEL09_ACTION_DATA_PREPARE_INT is called separately for each action to be performed in a LOOP (loop processing) via internal table ACTION_LT
1.3.1.3 Function Module VEO09_ACTION_DATA_PREPARE_INT

- Calling of PREPARE module for the appropriate action. Here a CASE statement is used to read the performing action and call the PREPARE module that has been assigned (hard-coded).
- Some PREPARE modules return the keys of template configurations in table VLCDGUIDOBJ_LT (see: 3.4.4.4). The configurations are read here so that they can be displayed later on the interface.
- PREPARE processing of customer-specific actions is called in method PREPARE_FURTHER_ACTIONS of BAdI PREPARE_FURTHER_ACTIONS (see: 3.4.5).
- Calling of module VEO09_PREPARE_VEHICLE_DATA

1.3.1.4 Action-Specific Module VEO<NN>_<ACTION>_PREPARE

Detailed description: see 3.4

1.3.1.5 Function Module VEO09_PREPARE_VEHICLE_DATA

If you have made settings in Customizing under the IMG activity Define Additional Data for Vehicle, this module adopts the task of reading and preparing the qualifier data and/or the vehicle attributes (see: 3.3).

In other words, this is a PREPARE module that can be defined as action-specific with the aid of Customizing settings.

1.3.2 EXECUTE Processing

EXECUTE processing represents the actual performing of an action. The business transaction is performed at this point by the writing of vehicle data, document data or other data to the database.

EXECUTE processing is started when the user has entered data and chosen the function for executing the action on the action screen (if required). It can also be started by calling a VMS BAPI, for example via an IDoc interface or the web interface.

If actions are performed via a VMS BAPI, only the EXECUTE modules are used. The checks implemented in the PREPARE modules have to be executed in the EXECUTE modules. In other words:

A action has to be completely described by its EXECUTE module, both logically and as regards the business side.
The following graphic shows which modules are called during EXECUTE processing. This general sequence is the same for all actions. The tasks of the individual modules are explained below.

Function module VEL09_SET_ACTION (also called “action handler”) is of particular importance. This is the central module for executing VMA actions; it is called both from the transaction VELO and from the VMS BAPIs.

Subroutine USER_COMMAND_4000 calls EXECUTE processing, i.e. module VEL09_SET_ACTION for action code AKTN (call action). This module then calls in turn module VEL09_SET_SINGLE_ACTION for each action to be performed (this can be more than one for interlinking actions). Then module VEL09_SET_ACTION_INTERNAL is called; this triggers the following functions:

- the action-specific EXECUTE module, if it exists
- method EXECUTE_FURTHER_ACTION for BADI VLC_EXECUTE_ACTION; the customer-specific EXECUTE modules are called in this BADI (see: 3.5.5)
- module VEL09_WRITE_VEHICLE_DATA If details (see: 3.3) have been defined in Customizing under the IMG activity Define Additional Data for Vehicle) to the effect that an action should prepare additional data (attributes or qualifiers), the additional data defined in Customizing is written from communication structure VLCACTDATA_CS to the vehicle data here.

© 2003 SAP AG
Neurottstr. 16
D-69190 Walldorf
Title: Vehicle Management System: Creating Actions
Version: 5.0
Date: 26.03.2003
The following graphic shows the data flow in EXECUTE processing:

EXECUTE processing receives the prepared data from the PREPARE module or the data entered by the user on the action screen (VLCACTDATA_S) and the list of vehicles to be processed (VLCDIAVEHI_T). When the action is performed, messages (error messages, warning messages, success messages) are collected in a message table and written to the action log (SAP re-use component: application log). Changes to the database are posted, documents are created and the results are displayed on the action screen.

1.3.2.1 Form USER_COMMAND_4000

Module USER_COMMAND_4000 has, amongst other things, the task of calling EXECUTE processing when an action is performed (function code AKTN). It fulfills the following tasks:

- Create an action log (module: VELO003_BAL_LOG_CREATE, re-use component)
- Fill vehicle table VLCDIAVEHI_LT: the vehicles that are currently in the vehicle list for tab page Action are listed in internal table VLCDISPLALV_ACTION_GT; the original of the vehicle data is in the global table VLCDISPLALV_ACTION_GT
- Transfer communication structure VLCACTDATA_CS from the global data of function group VEL009 (see 1.6.1); this means that all the data delivered by the PREPARE module and/or the action screens is known
- Call action handler (module: VEL009_SET_ACTION); amongst others, vehicle table VLCDIAVEHI_LT and communication structure VLCACTDATA_CS are transferred
- Transfer the messages returned by the action handler (error, warning, success messages) to the action log; if required, save log to database (controlled by user parameter VEL0_ACTPROTOCOL)
• Update some global (vehicle) tables and structures
• Output of a general error and/or success message
• Transfer communication structure VLCACTDATA_CS to VELO09 so that the action screens can display the most up-to-date data

1.3.2.2 Function Module VELO09_SET_ACTION
This function module – also known as the action handler – is the central module for executing VMS actions. It is called both from transaction VELO and from the VMS BAPIs. The following steps are performed:
• Set global VMS indicator in function group VELO22 (see: 1.6.3) This indicator can, for example, be read in a R/3 standard function customer exit in order to decide whether the function was called by VMS
• Check whether action exists/read action texts in the logon language
• Check whether the user is allowed to perform the action(s)
• Fill internal table ACTION_LT with the actions to be performed (this can be more than one for an interlinking action)
• Create vehicle (initial) when an action for creating vehicle data is called
• The following is observed per vehicle:
  • The system checks whether the user is authorized to perform an action for the plant.
  • The vehicle is locked for other users.
  • The system executes a Read-Before-Write-Check. This check compares the data displayed on the interface with the data current data in the database. If there are inconsistencies because another user has performed an action for the vehicle in the meantime, an error is triggered.
  • If the action is an interlinking action, the system checks whether it is allowed with the current vehicle status.

Depending on whether error occur or not, the individual vehicles are distributed to both internal tables VLCDIAVEHI_NO_LT and VLCDIAVEHI_OK_LT. Only the vehicles in the OK table are processed further.

• The following is performed separately for every action that is in internal table ACTION_LT:
  • The following is performed for every vehicle in the OK table:
    • Check whether the action is allowed for the current vehicle status.
    • Checks that are implemented in BAdI VLC_EXECUTE_ACTION are performed.

Depending on whether errors occur or not, vehicles from the OK table are written to the NO table. Vehicles in the NO table are defined as incorrect and are not processed further.
The following describes how the vehicles in the OK table are processed.

- **Special case:** if the action has been defined as an “isolating” action in Customizing (/SE16; table CVLC03; field ISOLTN = X), then the following steps are not performed, only method VEHICLE_ISOLATION of BAdI VEHICLE_ISOLATION is executed. There are already solutions for the implementation in customer projects.

  Vehicle “isolation” means that the action EXECUTE module is called separately for each vehicle. For example, if you perform the ordering action for 10 vehicles in “isolation” mode, one order is created automatically for each vehicle. It may be that 8 orders are successful and 2 are not created.

- **Call module VELO09_SET_SINGLE_ACTION; amongst other things, all the vehicle in the OK table VLCDIAVEHI_OK_LT and the data in communication structure VLCACTDATA_CS is transferred.**

- **When the action has been successfully completed, the COMMIT is triggered. A variable COMMIT_LV (or import parameter COMMIT_IV) decides whether the system posts the data synchronously or asynchronously.**

- **Actions are posted asynchronously in accordance with the default settings. Interlinking actions form an exception. Here, the system posts the data synchronously after each elementary action for safety reasons (COMMIT WORK AND WAIT). If you wish to deviate from this rule, you can implement BAdI VLC_EXECUTE_ACTION (method COMMIT_MODE).**

- Create success messages
- Update export parameters
- If an error occurs, ROLLBACK is triggered, error messages are generated, the export parameters are adjusted accordingly and the vehicles are unlocked.

- The loop ends via action table ACTION_LT.
- All vehicles are unlocked.

### 1.3.2.3 Function Module VELO09_SET_SINGLE_ACTION

This module receives a vehicle table (OK table), the communication structure and an action abbreviation and then executes the main steps of the action:

- **Call BAdIs for customer-specific changes to data before action is performed (BAdI VLC_EXECUTE_ACTION, method DATA_CHANGES_BEFORE_ACTION)**
- **Call module VELO09_SET_ACTION_INTERN; this calls the action-specific EXECUTE module**
- **Call BAdIs for customer-specific changes to data after action is performed (BAdI VLC_EXECUTE_ACTION, method DATA_CHANGES_AFTER_ACTION)**
- **Update vehicle location, availability and the planned delivery date in the vehicle data (IMG activity Define Action Controls and Define Action Matrices).**
- **Update vehicle status in the vehicle data**
- **Update the vehicle history**
1.3.2.4 Function Module VELO09_SET_ACTION_INTERN

- Calling of EXECUTE module for the appropriate action. Here a CASE statement is used to read the performing action and call the EXECUTE module that has been assigned (hard-coded).
- Customer-specific actions and modules are called in method EXECUTE_FURTHER_ACTIONS of BAdI VLC_EXECUTE_ACTION (see: 3.5).

The EXECUTE modules of customer-specific actions are called in method EXECUTE_FURTHER_ACTIONS of BAdI VLC_EXECUTE_ACTION.
- Calling of module VELO09_WRITE_VEHICLE_DATA.

1.3.2.5 Action-Specific Module VELO<NN>_<ACTION>_EXECUTE

Detailed description: see 3.5

1.3.2.6 Function Module VELO09_WRITE_VEHICLE_DATA

If you have made settings in Customizing under the IMG activity Define Additional Data for Vehicle, this module adopts the task of processing the qualifier data and/or the vehicle attributes (see: 3.3).

In other words, this is a EXECUTE module that can be defined as action-specific with the aid of Customizing settings.

1.3.3 Interlinking Actions: Process Flow

When the inter-linking action is started, all the other elementary actions contained in it are performed in the given order. The system processes each elementary action as a complete action, including setting a new status and updating data.

The interlinking of actions does not require any coding. However, if each of the individual elementary actions requires different entry fields, you have to create a new action screen that contains all of these fields and assign it to the interlinking action in Customizing under the IMG activity Define Technical Details for Actions (see: 2.2).

When you perform an interlinking action, the subscreen that has been assigned to the interlinking action in Customizing is processed. The screens for the elementary actions are not taken into account.

As described above, the PREPARE and EXECUTE modules are processed one after the other in interlinking actions.
When performing an interlinking action, PREPARE modules for all elementary actions are executed one after the other during PREPARE processing and all the relevant EXECUTE modules are executed one after the other during EXECUTE processing.

Example: if two actions A1 and A2, each consisting of one PREPARE module (P1 and P2), one EXECUTE module (E1 and E2) and one subscreen (S1 and S2) are linked to form one interlinking action A3 with its own subscreen S3, then the parts of the interlinking action A3 are performed in the following sequence: P1 → P2 → S3 → E1 → E2. Subscreens S1 and S2 are not required for A3.
1.4 Vehicle Table/Data Design

The most important VMS tables are discussed below. Please note, however, that links to other SAP standard tables (example: configuration, batch, valuation type, equipment…) exist.

- **Vehicle table VLCVEHICLE**
  
  This is the central vehicle table: it contains controlling and descriptive attributes. The structure VLCADDDDATA groups the descriptive vehicle attributes; this structure can be enhanced by customer-specific fields (see: 3.3.1).

- **Qualifier data for vehicle VLCADDDATA**
  
  You can define qualifiers in Customizing for descriptive vehicle data (see: 3.3.2), for example number plate, axle load, engine performance. These qualifiers are saved in this table with a defined vehicle reference.

- **Vehicle history VLCHISTORY**
  
  Every action that is performed is saved in the vehicle history with a time stamp, status transition and further data. This structure can also be enhanced by customer-specific fields (see: 3.3.1.1).

- **VMS document tables (for example VLCCUORDER, see: 3.5.4.4)**
  
  Many different SAP standard documents are created from VMS; there is a VMS document table for each document category in which the link between vehicle and document number/item is saved.

- **Additional vehicle configurations (VLCADDCFG)**
  
  A vehicle has exactly one vehicle configuration; this configuration is represented by a field in the vehicle table VLCVEHICLE-CUOBJ). A vehicle can have additional configurations in addition to the original vehicle configuration; these additional configurations are stored in table VLCADDCFG.
1.5 Communication Structure/Data Design

As already explained, communication structure VLCACTDATA_CS is responsible for the transport of data from PREPARE processing to the action screen and then from there on to EXECUTE processing.

You can only use fields from VLCACTDATA_CS on the action screens.

VLCACTDATA is structured as follows:

- **Header data** (structure VLCACTDATA_HEAD_S)
  
  Data that applies to all vehicles involved in the action. Example: if a sales order is created for three vehicles, the sold-to party is a header date.

  Structure VLCACTDATA_HEAD_S is divided into further substructures (see graphic; same as ITEM structure) in which the vehicle attributes are sorted thematically.

  Customer-specific fields should always be saved to one of the substructures. It may also make sense to include a new structure in VLCACTDATA_HEAD_S.

- **Item data** (table type VLCACTDATA_ITEM_T; field ADDDATA_ITEM)

  Data that applies to only one vehicle when the action is performed. Example: if action SMOD Change Vehicle is executed for several vehicles at the same time, only the vehicle identification number for the vehicle that is currently chosen is displayed on the action screen.

  If structure VLCACTDATA_ITEM_S has to be enhanced, the same rules apply as for VLCACTDATA_HEAD_S.
• **Qualifier data** (table type VLCADDDATA_ITEM_T; field ACTDATA_ITEM)

Data that can be freely defined in Customizing (see: 3.3.2) and, like the item data, only applies to one vehicle when the action is executed. In contrast to item data, qualifier data is displayed as an ALV list on the action screen; furthermore, it is written to a separate table VLCADDDATA in the database. This means that is does not overload the central vehicle table.

Since qualifier data can be freely defined in Customizing, no structure enhancements are required.
1.6 Global Data Objects in Action Processing

Several global data objects are used for action processing in transaction VELO. The main purpose of these objects is to enable data transport between different function groups and, above all, between many different screens (transaction VELO, action screens etc.).

When programming VMS, the number of global variables was kept to a minimum. Only function group VELO02 (transaction VELO) has many global variables. They are required here because of the numerous screens. The action function groups, for example VELO15, have only a few clearly-defined global data objects that are mainly required for the action screens. Furthermore, function group VELO09 (status and action management) has a few global data objects that are required in particular for the transfer of data between the function groups.

No global data objects are used in the VMS BAPIs or in function module VELO09_SET_ACTION. In other words: all data is transferred via the module interface. The same applies for all action-specific EXECUTE and PREPARE modules.

The global data objects of function groups VELO09, VELO22 and the action function groups are described below.

1.6.1 Global Variables in Function Group VELO09

This is the function group in which the general status and action management is implemented. The global data in this function group is required for the transfer of data between function group VELO02 (transaction VELO) and the different action function groups. VLCACTDATA_GS represents an exception.

- **VLCACTDATA_GS**
  Structure with action-specific data. This structure is filled by the fields on the action screen or by the modules VELO<NN>_<ACTION>_PREPARE. The information in the structure is required by modules VELO<NN>_<ACTION>_EXECUTE so an R/3 standard document, for example, can be created. Furthermore, return data for the action, for example the number of the document that is created, is returned to the action screen with this structure.

- **SELECTED_VEHICLE_GV**
  Internal number (GUID) of the vehicle chosen on the interface. This variable is required in order to be able to display item data belonging to the chosen vehicle on the action screen (for actions that contain item data).

- **LIST_OF_VEHICLES_GT**
  List of vehicles (corresponds to VLCDIAVEHI) that were chosen for the action. The list contains, amongst other things, vehicle data from table VLCVEHICLE.

- **CONFIGURE_BUTTON_GV**
  This variable contains information on whether the Configure function should be available on the action subscreen or not.

- **SELECTED_ACTION_GV**
Four-character abbreviation for the chosen action.

- **BEFORE_AFTER_GV**
  Variable for controlling the input/output fields and the pushbuttons on the action subscreen. This variable contains information for the action subscreen, whether the action is still to be executed or whether it has already been executed.

You can find further information in include LVELO09TOP.

### 1.6.2 Global Variables in Action Function Groups

There are some clearly-defined global data objects that are declared in all action function groups. The TOP include starts with the following lines for all action function groups:

```plaintext
FUNCTION-POOL VELOXX MESSAGE-ID VEO.
  *=================================================================================================================================================================

  * Vehicle Management System Type Pool
  TYPE-POOLS: VLCH,
  * IBase Type Pool
  IBXX.

  * The Include LVELO02CON contains all constants that are used in the
  * Vehicle Management System.
  INCLUDE LVELO02CON.

  * The Include LVELO15ACT contains declarations that are needed in all
  * function groups in that actions are implemented.
  INCLUDE LVELO15ACT.

  *=================================================================================================================================================================
```

- **LVELO02CON** is a central constant include in which (almost) all the constants used in VMS are defined. This include is part of VELO02 and is included in other function groups.

- There is a choice of global data objects that can basically be used in all action function groups. These are grouped together in include LVELO15ACT. This include is part of VEO15 and is included in all action function groups.
The following describes the global variables of include LVELO15ACT:

- **ACTIONTEXT_GV**
  
  Short text (language-dependent) for the action description. This is taken from Customizing and is displayed as a framed text on the action screen.

- **VLCACTDATA_HEAD_S**
  
  Data that applies to all vehicles involved in the action. This is usually header data for the documents that are to be created by the actions. The structure is part of VLCACTDATA and contains fields for all actions.

- **VLCACTDATA_ITEM_S**
  
  This is required only for actions that work with item data on the interface. It contains one line from table VLCACTDATA_ITEM_GT with the item data for the vehicle that is currently chosen.

- **VLCACTDATA_SUBITEM_S**
  
  This is only required for actions that work with subitem data on the interface. In other words, item data is used, but this data is not always part of a flat structure. It may have to be displayed as tables or via ALV. The structure represents one line of table VLCACTDATA_SUBITEM_GT and contains a subitem data record for the vehicle that is currently chosen.

- **VLCACTDATA_ITEM_GT**
  
  This is only required for actions that work with item data on the interface. It contains the item data for all the vehicles involved in the action. The table is part of VLCACTDATA.

- **VLCACTDATA_ADDDATA_ITEM_GT**
  
  This is only required for actions that work with additional data (qualifiers). It contains the additional data for all the vehicles involved in the action.

- **VLCACTDATA_VHCL_GT**
  
  This is only required for actions that work with additional data (qualifiers). It contains the additional data for the vehicle that is currently chosen.

- **VLCACTDATA_SUBITEM_GT**
  
  This is only required for actions that work with subitem data. It contains all the subitem data for all the vehicles involved in the action.

- **VLCACTDATA_VEHICLE_SUBITEM_GT**
  
  This is only required for actions that work with subitem data. It contains the subitem data for the vehicle that is currently chosen. Since there can be different types of subitem data for a vehicle, up to five global tables are supplied for this data.

- **CONTAINER_ADDDATA_GO; ALV_CONTROL_ADDDATA_GO**
  
  This is only required for actions that work with additional data (qualifiers). The qualifiers are displayed on the action screen by means of ALV. The variables are the object reference variables for the container and the ALV control.

- **VLC_SUBITEM_CONTAINER_1_GO, VLC_SUBITEM_GRID_1_GO**

© 2003 SAP AG  
Neuottstr. 16  
D-69190 Walldorf  

**Title:** Vehicle Management System: Creating Actions  
**Version:** 5.0  
**Date:** 26.03.2003  
Page 27 of 78
This is only required for actions that work with subitem data. The subitem data is displayed on the action screen by means of ALV. The variables are the object reference variables for the container and the ALV control. Since there can be different types of subitem data for a vehicle, these two variables are made available up to five times.

- **VLC_HTML_CONTAINER_GO; VLC_HTML_CONTROL_GO**
  This is only used for actions that use an HTML control on an action screen. The variables are the object reference variables for the container and the HTML control.

You can find further information in include LVELO15ACT. All the global variables for the action function groups are declared in this include. This include is used by all the action function groups.

### 1.6.3 Global Variables in Function Group VEL022

```plaintext
FUNCTION-POOL VEL022.

* This flag may only be set / read by the function modules VEL022_SET_
* VMS_FLAG and VEL022_GET_VMS_FLAG. The GET-function module can be
* called by R/3-standard documents to check whether the standard
* document is called by the Vehicle Management System or not. Depending
* on this information Modifications that are included in standard
* documents can be activated.

DATA: VMS_FLAG_GV TYPE XFLAG.

DATA: LIST_OF_VEHICLES_GT TYPE VLCDIAVEHI_T.
```

- **VMS_FLAG_GV**
  The VMS indicator is set automatically by the action handler (VELO09_SET_ ACTION). It can be read with module VEL022_GET_VMS_FLAG.

  Use: if you fill a customer exit for an SAP standard functionality with logic for your special VMS scenario, you can evaluate the VMS indicator in order to decide in an IF statement whether the coding for the customer exit should be run through or not.

- **LIST_OF_VEHICLES_GT**
  This vehicle list can be set and read with the modules VEL022_GET_LIST_OF_VEHICLES and VEL022_SET_LIST_OF_VEHICLES.

  Use: you can, for example, read the list of vehicles to be processed in a customer exit for an SAP standard functionality that is called by a VMS EXECUTE module. Before doing this you should, however, use the SET module to set the list in the EXECUTE module.
2 Creating Simple Actions

Simple actions are created solely with the aid of Customizing settings and do not require any ABAP programming knowledge.

2.1 Define Action

Perform the following steps in the IMG activity Define Actions (transaction OVELO1):

1. Create a key (four characters) and a description for each action.

2. Define whether the action should be an internal action.
   
   Internal actions are not available in transaction VELO but can still be executed via the VMS BAPIs (IDoc interface, web interface etc.).

3. Define whether the action should be a primary or a secondary action.
   
   A primary action is an action of primary action control and a secondary action is an action of secondary action control. Before executing the action, the system always checks the action matrix for the primary and/or secondary action control to decide whether this action is allowed on the basis of the vehicle status.

4. Define whether action SCEM should be relevant.
   
   Relevancy to SCEM determines whether the action that is performed should create an event handler in SCEM (Supply Chain Event Management) or change the information parameters of an existing event handler.

2.2 Entering Technical Data for Action

Perform the following steps in the IMG activity Define Technical Details for Actions (transaction OVELO6):

1. Define whether users should be able to configure data when they execute an action. There are the following options:
   
   - The user must configure the data: the configuration is started immediately and automatically. In other words, the IPC (Internet Pricing and Configuration) appears directly when the user calls the action.
   
   - The user is not allowed to configure data: the function for calling the configuration is not displayed.
   
   - The user may configure data: the function for calling the configuration is displayed.
2. Define whether the action should be a create action, i.e. whether it should, possibly in combination with other business functions, create vehicles.

Only the selected actions, i.e. only the actions for creating new vehicles are displayed on the vehicle search screen in transaction VELO since other actions are not required at this point.

It is on the basis of this indicator in function module VELO09_SET_ACTION that the system decides whether vehicles are created (initial).

3. Enter the program in which the action screen is contained and the screen number of the action screen.

4. Enter an alias action.

You can use the alias action to define for customer-specific actions which VMS standard action should be executed if the customer-specific action is chosen.

The use of simple actions for the purposes of configuration is not recommended. You may not set the indicator Create Action for simple actions. The fields Action Program and Action Screen also have to remain empty since simple actions do not use their own subscreens.

2.3 Define Vehicle Status

Here you can define the vehicle statuses that a vehicle passes through during its lifecycle.

The vehicle status changes automatically depending on the action you perform. We differentiate between primary and secondary statuses according to whether the status is referring to primary or secondary actions.

Create a key and a description for each status in the IMG activity Define Vehicle Status (transaction OVELO12).

You can also work with asterisks as wildcard characters and thus describe an entire cross-section of vehicle statuses. This is useful, for example, if you want to express in the action matrix that an action can not only be executed at one particular point in the process but also at a point within the entire cross-section of the process chain. For example, you can enter MM1* for MM10 to MM19, MM* for MM10 to MM99 or M*, *.
2.4 Assigning Actions to Action Matrices

You have to assign the action to at least a primary and/or a secondary matrix, depending on what has been defined in Customizing.

Perform the following steps in the IMG activity Define Action Controls and Define Action Matrices (transaction OVELO11):

- Action control is a key that represents an action matrix.

  1. Define a key and a description for each action control.
  2. Choose the action control to which you want the action to belong.
  3. Define the action matrix for this action control. In other words, define from which status the action should be executed and which status should be set after the action has been executed successfully.

     It could be that the action does not change the vehicle status. In this case, the old status and the new status are the same.

2.5 Define Action Control Determination

Then you have make settings for action control determination so that the system can assign a primary and/or secondary action control to the vehicles. At least one control must be found when the vehicle is created if the action is to be successful.

Enter the vehicle model in transaction VELOS and assign a primary and secondary action control to it.

2.6 Test Action

Now test the action you have created by creating a new vehicle.

- Only status changes are made for simple actions; you cannot make configuration changes. The action screen should therefore only contain the function for executing actions.

- You can track the appropriate status changes and historical data entries in the detail screen.
2.7 Linking Actions

You create interlinking actions in the same way as elementary actions (see: 2.1). Then assign the elementary actions to the interlinking action in the Interlinking Actions view and use a numerical value to define the sequence in which these actions should be executed.

- You can only create the interlinking action when you have both the authorization for the interlinking action and for all the elementary actions.
- The system processes each elementary action as a complete action in itself, including the setting of a new status and the updating of the data. It is for this reason that you have to enter each elementary action as an individual entry in the primary and/or secondary matrix. An intermediate status is therefore set by each elementary action when the interlinking action is processed.
- You also have to enter the interlinking action in the primary and/or secondary matrix. This entry is relevant due to the check on the old vehicle status, not due to the setting of a new status.
- Recursive interlinking is not possible. Interlinking actions may therefore not be used as elementary actions in interlinking actions.
3 Creating Complex Actions

First of all you have to create a simple action, as described above. Then you can build on it and program a complex action.

3.1 Creating Your Own Action Function Group

Create a function group. If you want this function group to be able to include actions, you have to declare several global data objects in the TOP include and include several PAI and PBO modules in the main program. To do this, copy the following lines to the TOP include of the new function group (see: 1.6.2):

```
FUNCTION-POOL VELOXX MESSAGE-ID VELO.

*=============================================================================

* Vehicle Management System Type Pool
TYPE-POOLS: VLCH,
* IBase Type Pool
    IBXX.

* The Include LVELO02CON contains all constants that are used in the
* Vehicle Management System.
INCLUDE LVELO02CON.

* The Include LVELO15ACT contains declarations that are needed in all
* function groups in that actions are implemented.
INCLUDE LVELO15ACT.

*=============================================================================
```
The PAI modules, PBO modules and subroutines you require are defined in function group VELO15. Include these objects in the main program of the new function group as follows:

* PBO : action screens (general)
* ================================================================
  INCLUDE LVEL015001.  " M_PREPARE_DATA (PBO)
  INCLUDE LVEL015F01.  " F_PREPARE_DATA (PBO)
  INCLUDE LVEL015002.  " M_DETERMINE_LAYOUT (PBO)
  INCLUDE LVEL015F02.  " F_DETERMINE_LAYOUT (PBO)
  INCLUDE LVEL015003.  " M_DETERMINE_VEHICLE_DISPLAY (PBO)
  INCLUDE LVEL015F03.  " F_DETERMINE_VEHICLE_DISPLAY (PBO)

* PAI : action screens (general)
* ================================================================
  INCLUDE LVEL015I01.  " M_CHECK_ENTRY_FIELDS_FILLED (PAI)
  INCLUDE LVEL015F04.  " F_CHECK_ENTRY_FIELDS_FILLED (PAI)
  INCLUDE LVEL015I02.  " M_MODIFY_ITAB_FROM_DISP (PAI)
  INCLUDE LVEL015F05.  " F_MODIFY_ITAB_FROM_DISP (PAI)
  INCLUDE LVEL015I03.  " M_TRANSFER_DATA (PAI)
  INCLUDE LVEL015F06.  " F_TRANSFER_DATA (PAI)

* PBO : action screens (with additional vehicle data table)
* ================================================================
  INCLUDE LVEL015004.  " M_ADDITIONAL_VEHICLE_DATA_PBO
  INCLUDE LVEL015F07.  " F_ADDITIONAL_VEHICLE_DATA_PBO

* PAI : action screens (with additional vehicle data table)
* ================================================================
  INCLUDE LVEL015I04.  " M_ADDITIONAL_VEHICLE_DATA_PAII
  INCLUDE LVEL015F08.  " F_ADDITIONAL_VEHICLE_DATA_PAII
3.2 Creating Action-Specific Subscreen

As a rule, complex actions have their own subscreen. The action screens have an identical structure (see: graphic) and a defined structure with respect to their modules and flow logic. Many actions require inbound parameters for processing. The user can enter these inbound parameters on an action-specific screen.

The following describes how you can include additional fields on action screens and process them in the course of action execution. You can use this method both to enhance existing action screens and to create your own action screens.

Even if you only make small changes to existing action screens, you should not modify them. You should copy them to a separate function group and then change the copy there.

3.2.1 Screen Elements/Layout

SAP recommends that you always copy an existing screen and then adapt the copy so that the names and attributes of the screen elements coincide for all VMS actions.

Frame Name
ACTIONTEXT_GV

Fields have to be part of:
VLCACTDATA_ITEM_S
and/or
VLCACTDATA_HEAD_S

Screen Painter fields:
IN1 (required), IN2 (optional), OU1 (output)

Buttons (Screen Painter):
BUTTON_KONF → Start IPC
BUTTON_ADDR → Maintain end customer
BUTTON_AKTN → Execute action
• **Frame**

The frame of the action-specific screen must be called ACTIONTEXT_GV. Only then can you be sure that the text defined in VMS Customizing appears automatically for the action description on the action screen.

• **Fields**

All the fields used on the action screens have to be part of structure VLCACTDATA_ITEM_S (item data) or structure VLCACTDATA_HEAD_S (header data). This ensures that they are automatically part of communication structure VLCACTDATA_CS so that the transfer of data from the PREPARE module via the action screen to the EXECUTE module functions correctly.

• **Modification Groups**

The following modification groups are handled automatically by PBO module M_DETERMINE_LAYOUT (see: 3.2.2.1) in the way described below. Assign the screen fields to the appropriate modification groups:

- **IN1**: required entry fields for action – only ready for input before the action is executed
- **IN2**: optional entry fields for action – only ready for input before the action is executed
- **OU1**: output fields – only visible after the action has been executed. Fields of this modification group have to be defined on the subscreen as not visible and as only an output field.

• **Pushbuttons**

- **BUTTON_KONF**
  
  Calls IPC (function code KONF); is not displayed after the action has been executed.
  
  This pushbutton can be hidden in accordance with settings in Customizing for the actions (see: 2.2); PBO module M_DETERMINE_LAYOUT is responsible for the control of this (see: 3.2.2.1).

- **BUTTON_AKTN**
  
  Starts action execution (function code AKTN); has to be used by each action screen; is hidden after the action has been executed.

- **BUTTON_ADDR**
  
  Calls central address management (function code ADDR); is not used on every action screen; is hidden after the action has been executed.

• **Custom Control** (for qualifiers, see 3.3.2)

If you want to work with vehicle qualifiers, you have to add a custom control with the technical name ADDDATA_CONTAINER on the action screen. Action screen 100 in function group VELO15 (action SMOD Change Vehicle) can be used as a template.
3.2.2 Flow Logic

SAP recommends that you only use the standard modules as you are then making use of maintenance-free action screens. The options offered to you by these modules are usually sufficient. The existing modules have to be run through in a predefined order in the flow logic:

* Action XXXX (text)
*=================================================================================================

PROCESS BEFORE OUTPUT.

* Get VLCACTDATA to fill the fields of the subscreen
  MODULE M_PREPARE_DATA.

* Find out which buttons / fields / ... should be visible
  MODULE M_DETERMINE_LAYOUT.

* Determine the line of VLCACTDATA_ITEM_GT which has to be displayed now
  MODULE M_DETERMINE_VEHICLE_DISPLAY.

* Prepare and display the alv-list and fill VLCADDDATA_VHCL_GT
  MODULE M_ADDITIONAL_VEHICLE_DATA_PBO.
  *=================================================================================================

PROCESS AFTER INPUT.

CHAIN.
  FIELD VLCACTDATA_ITEM_S-VHVIN.
  FIELD VLCACTDATA_ITEM_S-PDDATU.
  FIELD VLCACTDATA_ITEM_S-VHUSG.
  FIELD VLCACTDATA_ITEM_S-VHSDAR.
  FIELD VLCACTDATA_ITEM_S-SHVL.
* Find out if all mandatory entry fields have been filled by the user
  MODULE M_CHECK_ENTRY_FIELDS_FILLED.
ENDCHAIN.

* Put the modified data of the previously selected vehicle back to table
  VLCACTDATA_ITEM_GT
  MODULE M_MODIFY_ITAB_FROM_DISP.

* Check data entered in the alv-list and fill VLCADDDATA_ITEM_GT
  MODULE M_ADDITIONAL_VEHICLE_DATA_PAI.

* Fill VLCACTDATA using the data entered on the subscreen
  MODULE M_TRANSFER_DATA.
  *=================================================================================================

According to whether the action screen works with header data, item data and/or qualifier data (see: 1.5), certain modules are required or not (see below).
You must list the entry fields in the CHAIN statement. This is important so that the field is ready for input again after incorrect data has been entered.

### 3.2.2.1 Modules in Process Before Output

The following modules can be found in function group VEL015 and are included in their own action function groups:

- **M_PREPARE_DATA**
  
  This is used for all actions. It is processed once when an action is called. The task of this module is to bring data that is read from VELO15_<ACTION>_PREPARE modules to VLCACTDATA onto the interface by filling VLCACTDATA_HEAD_S and VLCACTDATA_ITEM_GT.

- **M DETERMINE_LAYOUT**
  
  This is used for all actions. It serves to control the display of the action-specific subscreens in accordance with the variables BEFORE_AFTER_GV. For example, you can control whether entry fields are hidden or displayed, ready for input or not and also whether pushbuttons are displayed with this module. Entry fields, for example, are no longer ready for input after the action has been executed.

  You can control whether the **Configure** pushbutton (BUTTON_KONF) is displayed or hidden by using settings in Customizing for the actions (see: 2.2).

- **M DETERMINE_VEHICLE_DISPLAY**
  
  This is only used for actions for which there is item data as well as header data. In other words, data that is different for the vehicles involved in the action. This module ensures that the vehicle data, which has been chosen in the vehicle list, is displayed in the item detail part of the subscreen. Data from internal table VLCACTDATA_ITEM_GT is written to structure VLCACTDATA_ITEM_S. The number of the chosen vehicle is read from global variable SELECTED_VEHICLE_GV.

- **M_ADDITIONAL_VEHICLE_DATA_PBO**
  
  This is only required for actions with qualifier data. It serves to write the additional data of the vehicle that has been selected in the vehicle list to the internal table and display the contents of this table on the action screen (ALV).

### 3.2.2.2 Modules in Process After Input

The following modules can be found in function group VEL015 and are included in their own action function groups:

- **M CHECK_ENTRY_FIELDS_FILLED**
  
  Checks whether all the entry fields of modification group IN1 (required entry fields) have been filled when the user has chosen the function for executing the action. Only then will the action be executed.
• **M_MODIFY_ITAB_FROM_DISP**
  
  This is only used for actions for which there is item data as well as header data and represents the counterpart of M_DETERMINE_VEHICLE_DISPLAY. The item data entered on the subscreen are used to update the data of the selected vehicle in the internal table of the vehicles relevant to the action.

• **M_TRANSFER_DATA**
  
  When the user has chosen the function for executing the action, internal table VLCACTDATA_ITEM_GT and structure VLCACTDATA_HEAD_S are written to VLCACTDATA. The transfer of data in VLCACTDATA to the VEO<NN>_<ACTION>_EXECUTE modules is secured automatically.

• **M_ADDITIONAL_VEHICLE_DATA_PAI**
  
  This is only required for actions with qualifier data. It writes the internal additional data table for the vehicle that is displayed back to the internal additional data table for all vehicles.

### 3.2.3 Assigning Action Screens to Actions

Assign the action screen you have created to the action (see: 2.2).
3.3 Processing Customer-Specific Additional Data

There are at least two techniques for processing and saving customer-specific additional data in the Vehicle Management System:

- **Vehicle Attributes**
- **Qualifiers**

The vehicle attributes technique requires you to invest a little time and effort in programming; the qualifier technique requires no additional programming.

3.3.1 Processing Customer-Specific Additional Data as Vehicle Attributes

Vehicle attributes are stored in the central vehicle table VLCVEHICLE; they are fields in this table.

You can decide whether an attribute is valid for all the vehicles involved when an action is processed or whether the attribute has vehicle-specific data.

You have to define new customer-specific attributes as data elements and copy them to vehicle table VLCVEHICLE and communication structure VLCACTDATA.

3.3.1.1 Enhancing Structure VLCACTDATA

The structure of VLCACTDATA has already been explained (see: 1.5). Depending on whether an attribute should be valid for all vehicles when an action is processed or whether it should only contain vehicle-specific data, you have to enhance either VLCACTDATA_HEAD_S or VLCACTDATA_ITEM_S.

- **Header data** (structure VLCACTDATA_HEAD_S)
  
  Data that applies to all vehicles involved in the action. Example: if a sales order is created for three vehicles, the sold-to party is a header date.

  Structure VLCACTDATA_HEAD_S is divided into further substructures (see: 1.5) in which the vehicle attributes are sorted thematically.

  You should always enhance one of the substructures for customer-specific fields. It may also make sense to include a new structure in VLCACTDATA_HEAD_S.

- **Item data** (table type VLCACTDATA_ITEM_T; field ADDDATA_ITEM)

  Data that applies to only one vehicle when the action is performed. Example: if action SMOD Change Vehicle is executed for several vehicles at the same time, only the vehicle identification number for the vehicle that is currently chosen is displayed on the action screen.
If you want to enhance structure VLCACTDATA_ITEM_S, the same rules apply as for VLCACTDATA_HEAD_S. You have to copy the new attributes to vehicle table VLCVEHICLE by using the APPEND technique. It is important that the field names in the vehicle table and VLCACTDATA are the same.

You can also add attributes to the vehicle history table VLCHISTORY. If the field names coincide with the field names in the vehicle table, the vehicle history is automatically filled with the appropriate data when the action is processed.

### 3.3.1.2 Assigning Vehicle Attributes to Actions

You can assign vehicle attributes to actions in the IMG activity *Define Additional Data for Vehicle*. This results in the data being processed automatically when the action is executed.

Perform the following steps in this IMG activity:

1. Select the desired elementary action.
2. Enter the vehicle attributes for the elementary action.
   - Before doing this, you must have entered the new vehicle attributes in control table VLCVALIDFIELDS (transaction VELO_SA).
3. Define whether you are dealing with a header field or an item field.

   **Header field** means that the field is copied from VLCACTDATA_HEAD_S to the vehicle data for all involved vehicles when the action is executed. This process is, of course, only successful if the field exists both in VLCACTDATA_HEAD_S and in vehicle table VLCVEHICLE.

   **Item field** means that the field is copied from VLCACTDATA_ITEM_S to the vehicle data of the appropriate vehicle when the action is executed. A separate data record exists for each vehicle involved. This process is, of course, only successful if the field exists both in VLCACTDATA_ITEM_S and in vehicle table VLCVEHICLE.

You have to do some of your own programming (see: 3.2) if you want to be able to display and maintain the data on the action screen. The action screen must contain the fields from VLCACTDATA; the modules described are required in the workflow. If you use item data, you have to integrate the modules M_DETERMINE_VEHICLE_DISPLAY and M_MODIFY_ITAB_FROM_DISP.
3.3.2 Processing Customer-Specific Additional Data as Qualifiers

Qualifiers are freely-definable in Customizing. They carry vehicle-specific data that is written to the database of a separate table VLCADDDATA when the action is executed. Therefore, they do not overload the central vehicle table.

They are displayed as ALV list on the action screens.

Since qualifier data can be freely defined in Customizing, you do not have to make any structure enhancements.

3.3.2.1 Defining Qualifier Data

Create a key and a description for each qualifier in the IMG activity Define Additional Data for Vehicle.

3.3.2.2 Assigning Qualifier Data to Actions

You can assign qualifiers to actions in the IMG activity Define Additional Data for Vehicle. This results in the qualifier data being processed automatically when the action is executed.

Perform the following steps in this IMG activity:

1. Select the desired elementary action.
2. Enter the qualifier data for the elementary action.
3. Choose the desired maintenance type (required, optional, display).

You have to make some programming amendments if you want to be able to display/maintain the qualifier data on the action screen (see: 3.2). The action screen must contain a custom control with the technical name ADDDATA_CONTAINER in the layout.

You have to integrate both modules M_ADDITIONAL_VEHICLE_DATA_PBO and M_ADDITIONAL_VEHICLE_DATA_PA1 in the flow logic.
3.3.3 Qualifier Data Versus Vehicle Attributes

It makes sense to work with attributes and qualifiers if you require additional data for a vehicle and this additional data cannot be covered by characteristics (for example: license plate number) and if you want to make this information available on action screens and in the vehicle search.

From a business point of view, there is no difference between using qualifier data or vehicle attributes. Both concepts make parameters with additional customer-specific information for a vehicle available.

From a technical point of view, there are the following differences:

- **Attributes** are written directly to the vehicle table VLCVEHICLE. This increases the scope of the vehicle table, which may have an effect on performance when the actions are executed. Attributes are displayed as separate additional fields on the action screen.

- **Qualifier data** is managed in its own additional table VLCADDDATA. The scope of the vehicle table remains unaltered and so does performance. Qualifier data is displayed on the action screen in a separate area and as an ALV list.

Use **qualifier data** to store additional data for a vehicle when the additional data fulfills the following criteria:

- The data is descriptive in nature, not controlling data.
- The number of data fields you require is relatively large.
- The data is only required for a few actions.

If these criteria do not apply, you should use **vehicle attributes**.
3.4 Creating and Calling a PREPARE Module

Not every action requires its own PREPARE module. Action-specific PREPARE modules have the following tasks:

- Data is read and prepared in order to transfer it to communication structure VLCACTDATA_CS and display it as default value or information on the action screen.
- The PREPARE module is called directly after the action to be executed has been chosen in transaction VELO.

The system can check at the start of action processing whether the action can be executed on the grounds of the current data constellation. A termination at this point is more user-friendly than waiting until EXECUTE processing (after the user has entered data on the action screen and chosen the function for executing the action).

Examples for a check:

- Action CHCO (Change Sales Order) checks whether a sales order exists for the vehicle. If not, processing is terminated.
- Action OUVI (Create Outgoing Invoice) checks whether a sales order exists as a preceding document. If not, processing is terminated.

First of all you have to create a customer-specific action function group (see: 3.1) to which you can add the PREPARE module, the EXECUTE module and the action screen.

3.4.1 Naming Conventions

Please adhere to the following naming conventions for PREPARE modules:

<Name of function group>_<Abbreviation of action>_PREPARE

3.4.2 Documentation

The standard action modules sometimes have the same or similar commentary blocks in their coding. Use a method of commenting that is consistent, even for your own action modules, in order to achieve the greatest possible transparency for everyone.
3.4.3 Interface

The interface for the PREPARE modules has been relatively predefined. Simply copy the standard module (look in the action function groups – see: 1.2). By doing this, you also copy the general structure and the commentary blocks as well as the interface.

```
FUNCTION_VEL010_MORD_PREPARE.
**-------------------------------------------------------------
*** Lokale Schnittstelle:
** IMPORTING
**  REFERENCE(XINTERLINKED_IV) LIKE CVLC03-INTRLK OPTIONAL
**  REFERENCE(XCALLEDBYEXECUTE_IV) TYPE VLC_XCALLEDBYEXECUTE
**  OPTIONAL
** TABLES
**  VLCDIAVEHI_IT STRUCTURE VLCDIAVEHI OPTIONAL
**  VLCGUIDCUOBJ_ET STRUCTURE VLCGUIDCUOBJ OPTIONAL
**  CHANGING
**  REFERENCE(VLCACTDATA_CS) TYPE VLCACTDATA
** EXCEPTIONS
**  VEHICLES_OF_SEVERAL_ORDERS
**  NO_ORDERS_FOUND_IN_VLC
**  ERROR_IN_BAPI_PO_GETDETAIL
**  NOT_PERFORMED
**-------------------------------------------------------------
```

- **XINTERLINKED_IV** (optional)
  This flag has a value of X if the action is executed as an elementary action within an interlinking action. You can read the flag in coding and use an IF statement in order to switch off a check for interlinking actions, for example, that is to be executed later in the EXECUTE module. This is useful, for example, when the system checks to see whether a preceding document exists.

  Example of interlinking of actions CUOR Create Sales Order and OUVI Create Outgoing Invoice:

  The action-specific PREPARE modules of both actions are run through one after the other when the PREPARE is called (see: 1.3.3). In the PREPARE module of the outgoing invoice, the system checks whether a preceding document, i.e. a sales order, exists. The sales order does not yet exist, of course, in an interlinking action; this means that an error message and the ensuing termination of the action processing has to be suppressed.

- **XCALLEDBYEXECUTE_IV** (optional)
  This flag has the value X when the PREPARE module is called by the EXECUTE module.

  Background: the checks that are made in the PREAPE module have to be repeated in the EXECUTE module (see: 3.5.4.2.2). It could therefore make sense to call the PREPARE module again in the EXECUTE module.

  You can read this flag in Customizing in order to be able to react less tolerantly towards the non-existence of preceding documents when calling from the EXECUTE module, for example.

- **VLCDIAVEHI_IT** (required)
The contents of this table serve merely an informative purpose and may not be changed in the PREPARE module.

- VLCUIDCUOBJ_ET (optional)
  Export table in which the template configuration is defined. This template configuration is displayed in the IPC (Internet Pricing and Configuration) and can be changed by the user.

  The PREPARE module defined the template configuration. Example: in action CUOR Create Sales Order, the configuration of a quotation vehicle that possibly exists is used; if no quotation exists, the vehicle configuration is used as a template.

- VLCACTDATA_CS (necessary)
  This is a communication structure that supports the data flow from the PREPARE module via the action screen to the EXECUTE module.

  Data, for example, that is to be displayed on the action screen or that is to be transferred to the EXECUTE module is written to this structure.

  Because of the existence of interlinking actions, a PREPARE module may NEVER delete the content of this structure, it should only ever add to it.

- Exceptions (mostly necessary)
  There is a basic rule in VMS programming: error messages should not be triggered within a function module; they should rather be handed over step by step to the interface. This means:

  When an error occurs in a PREPARE module, the error message is transferred to the calling program by using a RAISE statement (coding example: MESSAGE E001 RAISING NOT_PERFORMED).
3.4.4 Source Text

Implement the PREPARE module basically in such a way so that it could be interlinked. Please note the comments on variables XINTERLINKED_IV (see: 3.4.2) and implement the basic rule that communication structure VLCACTDATA_CS may never be deleted, data may only be added to it. The refresh is executed at a central point.

Use the option of copying source text from the standard actions. This is quick and easy and ensures data consistency.

Copy the elements that you require for you case from the function modules of the standard action function groups VEO10, VEO11 and VEO15 etc.

You can find some examples of coding below:

3.4.4.1 Data Declarations

It is helpful if the same variables are also called the same in the different PREPARE modules. Therefore please also observe the same rule here: copying is quicker and ensures consistency in the logic.

```plaintext
**---------------------------------------------------------------
* This function module reads checks and prepares data needed for the *
* action XYZW before the subscreen of the action can be displayed. The *
* function module fills structure VLCACTDATA_CS with action specific *
* data. This function module must be processed before the PBO of the *
* action's subscreen is processed. *
**---------------------------------------------------------------

* Declarations
* =========

* Vehicles selected for XYZW
DATA: VLCDAVEHI_LS TYPE VLCDAVEHI,

* VLCACTDATA
  VLCACTDATA_ITEM_BEFORE_LT TYPE VLCACTDATA_ITEM_T,
  VLCACTDATA_ITEM_AFTER_LT TYPE VLCACTDATA_ITEM_T,
  VLCACTDATA_ITEM_LS TYPE VLCACTDATA_ITEM_S,

* Local structure / table for the cust. quotation table of the VMS
  VLCOFFER_LS LIKE VLCOFFER,
  VLCOFFER_LT LIKE TABLE OF VLCOFFER,

* GUIDs of the selected vehicles (for order selection)
  VLCGUID_LT LIKE TABLE OF VLCGUID,
  VLCGUID_LS LIKE TABLE OF VLCGUID,

* Other variables
  ...
```
- **VLCDIAVEHI_LS**
  Local vehicle structure that is required when vehicle table VLCDIAVEHI_IT (interface, see: 3.4.3) has to be read.

- **VLCACTDATA_ITEM**
  Data objects that are required when you work with item data (see: 1.5) from the communication structure.

- **VLCOFFER_LT**
  VMS document table for the standard document *Quotation*. There are separate VMS tables for the different document categories (see: 4.1).
  The relationship between vehicle and document number/item is stored in the document tables. If you know the vehicle, you can determine the appropriate documents in these tables.

- **VLCGUID_LT**
  GUID (globally unique identifier), in other words the key from vehicle table VLCVEHICLE that uniquely identifies a vehicle.
  Here an internal table is declared that can be filled with the GUIDs of the appropriate vehicles in order to select data from different tables.

- **Other variables**
  It is mostly simple and unstructured variables that are declared here.
3.4.4.2 Checks

One of the main tasks of the PREPARE modules is performing data checks.
Here are two examples:

3.4.4.2.1 Reading and Checking Preceding Document

*----------------------------------------------------------
* Step 1
* Fill table of GUIDs (needed as import parameter for
* VEL004_READ_SEVERAL_OFFER)

LOOP AT VLCDIAVEHI_IT INTO VLCDIAVEHI_LS.
  CLEAR: VLCGUID_LS.
  VLCGUID_LS-VGUID = VLCDIAVEHI_LS-VGUID.
  APPEND VLCGUID_LS TO VLCGUID_LT.
ENDLOOP.  "VLCDIAVEHI_IT

*----------------------------------------------------------
* Step 2
* Verify that the customer quotation exists for the selected vehicles by
* reading the Customer Quotation table of the Vehicle Manager

CALL FUNCTION 'VEL004_READ_SEVERAL_OFFER'
  EXPORTING
    VLCACTDOCTYPE_IV = ABA_OFFE_GC
  TABLES
    VLCGUID_IT = VLCGUID_LT
    VLCOFFER_ET = VLCOFFER_LT
  EXCEPTIONS
    NO_OFFER_FOUND = 1
    OTHERS = 2.

IF SY-SUBRC <> 0.
  MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4
       RAISING NO_CUSTOMER_QUOTATION_FOUND.
ENDIF.

*----------------------------------------------------------
• Step 1
  Internal table VLCGUID_LT is filled with the GUIDs of the appropriate vehicles.

• Step 2
  All the quotations (document number/item) for the vehicles affected are read in VMS
document table VLCOFFER.

In addition, selections are made using the **action document category**. The action
document category was introduced so that a separate VMS document table does not have
to be created for each document category (field ACTDOCTYPE in VMS document tables).

For example, both sales orders and returns are entered in the VMS table VLCCUORDER.
Technically speaking, both of these document categories are stored in the same table in
R/3 as well. You can usually use the action abbreviation as action document category.

You must always take the action document category (field ACTDOCTYPE) into
account when you write and read data in the VMS document tables; the action
abbreviation of the standard action is usually used as action document category.

Error handling: If an error occurs, the error message is not triggered here but is transferred
to the calling program by use of the RAISING statement.
3.4.4.2.2 Checks

Below is an excerpt from the coding for action GOIS Post Goods Issue. If you want to perform a goods issue, a delivery is required as preceding document:

```
* Read the delivery numbers for the selected vehicles from the delivery table of the Vehicle Management System
* (Function module VEO14_READ_DELIVERIES replaces VEO04_READ_DELIVERY)
CALL FUNCTION 'VEO14_READ_DELIVERIES'
  EXPORTING
    ACTDOCTYPE_IV = ABA_DELIVDOC_LVL
    LATEST_IV = 'X' " flag added for CRT4
  TABLES
    VGUIDS_IT = VLCGUID_LT
    ACTDOCTYPE_IT =
    DELIS_ET = VLCDELIVERY_LT
  EXCEPTIONS
    NO_ENTRIES_FOUND = 1
    NO_DATA_RECEIVED = 2
    OTHERS = 3.
* Error handling
  IF SY-SUBRC <> 0.
    IF NOT XINTERLINKED_IV IS INITIAL
       AND XCALLEDBYEXECUTE_IV IS INITIAL.
       EXIT.
     * The existing of the delivery is not mandatory if the action is performed as an interlinked action and if the calling function module is not the EXECUTE function module. In this case the processing of this function module is stopped.
    ELSE.
      Error: no delivery found.
      MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4
        RAISING DELIVERY_NOT_FOUND1.
      ENDF.
      "IF XINTERLINKED_IV IS INITIAL.
    ENDF.
* "SY-SUBRC
```

- Function module VEO14_READ_DELIVERIES reads VLCDELIVERY in the VMS document table to determine whether deliveries exist as preceding documents.

- If no deliveries exist, this is an error situation. Parameters XINTERLINKED_IV and XCALLEDBYEXECUTE_IV are used to decide whether the error should be suppressed at this point in time (background: see 3.4.2).
  - If there is no interlinking (XINTERLINKED_IV IS INITIAL), the error occurs invariably.
  - If the PREPARE module was called by the EXECUTE module (NOT XCALLEDBYEXECUTE_IV IS INITIAL), the error also occurs invariably.
3.4.4.3 Filling the Communication Structure

The PREPARE module should, on no account, delete VLCACTDATA_CS, otherwise you cannot interlink actions. You may, however, add data.

3.4.4.3.1 Header Data

Data from the appropriate sales order is written to the header segment of the communication structure in the PREPARE module of action CHCO Change Sales Order so that it can be displayed on the action screen.

* In order to display the data on the action subscreen, the data will be
* moved to the structure VLCACTDATA_CS.

VLCACTDATA_CS-KUNNR = VBAK_LS-KUNNR.
VLCACTDATA_CS-KUNAM = KUNAM_LV.
VLCACTDATA_CS-VBELN = VBELN_LV.
VLCACTDATA_CS-AUART = VBAK_LS-AUART.
VLCACTDATA_CS-VKORG = VBAK_LS-VKORG.
VLCACTDATA_CS-VTMG = VBAK_LS-VMG.
VLCACTDATA_CS-SPART = VBAK_LS-SPART.
VLCACTDATA_CS-BSTNK = VBAK_LS-BSTNK.
3.4.4.3.2  Item Data

* PREPARE table containing existing old data of VLCACDATA
VLCACDATA_ITEM_BEFORE_LT[] = VLCACDATA_CS-ACTDATA_ITEM.

* One line per vehicle ...
  LOOP AT VLCDIAVEHI_CT INTO VLCDIAVEHI_LS.

* Don't mix data of different vehicles
  CLEAR VLCACDATA_ITEM_LS.

* Use existing old data of VLCACDATA_CS (This read does only find
  entries in case of interlinked actions)
  READ TABLE VLCACDATA_ITEM_BEFORE_LT INTO VLCACDATA_ITEM_LS
  WITH KEY VGUID = VLCDIAVEHI_LS-VGUID.

* Write GUID into the new VLCACDATA (needed if VLCACDATA_ITEM_LS is
  initial)
  VLCACDATA_ITEM_LS-VGUID = VLCDIAVEHI_LS-VGUID.

* Add action-specific data to VLCACDATA_ITEM_LS

* Add line to new VLCACDATA
  APPEND VLCACDATA_ITEM_LS TO VLCACDATA_ITEM_AFTER_LT.

  ENDLOOP.  " VLCDIAVEHI CT INTO VLCDIAVEHI LS

* Replace old VLCACDATA by new VLCACDATA. Contains the entries of the
* old VLCACDATA and in addition data of the action.
  CLEAR VLCACDATA_CS-ACTDATA_ITEM.
  VLCACDATA_CS-ACTDATA_ITEM[] = VLCACDATA_ITEM_AFTER_LT[].

* First of all, field VLCACDATA_CS-ACTDATA_ITEM (contains item data) is written to the
  BEFORE table.
* If it exists, the line that fits the current vehicle is read from this old VLCACDATA version.
  The action-specific data (in particular the GUID) is added to this line and then appended to the
  AFTER table.
* Finally, field VLCACDATA_CS-ACTDATA_ITEM is deleted and overwritten by the AFTER
  table.

This procedure ensures that the table is not updated but merely enhanced with additional new
data. This is of particular importance when you are dealing with interlinking actions.
3.4.4.4 Defining a Template Configuration

You can define a template configuration in the PREPARE module. This configuration is displayed in the IPC (Internet Pricing and Configuration) and can be changed by the user.

*------------------------------------------------------------------
  IF VLGUIDCUOBJ_ET IS REQUESTED.
  * Provide a table containing a list of GUIDs and the CUOBJs of the
  * action's template configurations (in this case, the vehicle cfg is
  * the template cfg)
  LOOP AT VLCDIAVEHI_IT INTO VLCDIAVEHI_LS.
    VLGUIDCUOBJ_LS-VGUID = VLCDIAVEHI_LS-VGUID.
  * take the vehicle configuration
    VLGUIDCUOBJ_LS-CUOBJ = VLCDIAVEHI_LS-CUOBJ.
  * Add template configuration's CUOBJ to the list
    APPEND VLGUIDCUOBJ_LS TO VLGUIDCUOBJ_ET.
  CLEAR VLGUIDCUOBJ_LS.
  ENDLOOP.   " LOOP AT VLCDIAVEHI_IT INTO VLCDIAVEHI_LS
ENDIF.   " IF VLGUIDCUOBJ_ET IS REQUESTED.
*------------------------------------------------------------------

- In table VLGUIDCUOBJ_ET you can define per vehicle which configuration you want to use as the template configuration.
  Background: the configuration is stored in the IBase in R/3; there is a key (CUOBJ) that uniquely represents a concrete configuration.
  The vehicle configuration is represented in the coding by field VLCDIAVEHI_LS-CUOBJ.
  This field represents field VLCVEHICLE-CUOBJ in the central vehicle table VLCVEHICLE.
- The relationship between vehicle (GUID) and configuration (CUOBJ) is stored in internal table VLGUIDCUOBJ_ET.
- Table VLGUIDCUOBJ_ET is exported from the action-specific PREPARE module and then used centrally.
3.4.5 Call

Create an implementation for BAdI VLC_PREPARE_ACTION in transaction SE18. The action-specific PREPARE module is called in BAdI method PREPARE_FURTHER_ACTIONS.

Example of coding:

```
*************** CASE - LOOP on all actions **************
CASE ACTION_TO_BE_PERFORMED_IV.

WHEN '<Aktionskürzel1>' . " Customer's first action !
  '__________________ ACTION: <Aktionskürzel> ___________________

  CALL FUNCTION '<FugrName>_<Aktionskürzel1>_PREPARE'
    EXPORTING
      XINTERLINKED_IV  = INCOMING_ACTION_IS-INTRLK
      TABLES
        VLCODA_VEHIT    = LIST_OF_VEHICLES_IT
        VLCODC_OBJET    = VLCODC_OBJET
      CHANGING
        VLCACTDATA_CS  = VLCACTDATA_CS
    EXCEPTIONS
      NOT_PERFORMED  = 1
      OTHERS         = 2 .

  IF SY-SUBRC <> 0 .
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4
    RAISING ACTION_PREPARE_NOT_PERFORMED .
  ENDF.

WHEN '<Aktionskürzel2>' . " Customer's second action !
  '__________________ ACTION: <Aktionskürzel2> ___________________

    ... usw.

ENDCASE .
```
3.5 Creating and Calling an EXECUTE Module

The EXECUTE module is called when the user chooses the function for executing an action. The business process step is performed for all the vehicles involved and data and, where appropriate documents, are written to the database.

Only EXECUTE modules are run through for actions that are executed via VMS BAPIs. The checks implemented in the PREPARE modules have to be executed in the EXECUTE modules. In other words:

🎉 An Action has to be completely described by its EXECUTE module, both logically and as regards the business side.

Action-specific EXECUTE modules have, amongst other things, the following tasks:

- Check whether the action can be executed for the vehicles transferred on the basis of the current data constellation.
- Write data from communication structure VLCACTDATA_CS to the vehicle data.
- If necessary, call VMS BAPIs in order, for example, to create, change or delete SAP standard documents.
- Update VMS document table, if required.

3.5.1 Naming Conventions

Please adhere to the following naming conventions for PREPARE modules:

```
<Function group name>_<Action abbreviation>_EXECUTE
```

3.5.2 Documentation

The standard action modules sometimes have the same or similar commentary blocks in their coding. Use a method of commenting that is consistent, even for your own action modules, in order to achieve the greatest possible transparency for everyone.
3.5.3 Interface

The interface for the EXECUTE modules has been relatively predefined. Simply copy the standard module (look in the action function groups – see: 1.2). By doing this, you also copy the general structure and the commentary blocks as well as the interface.

```
FUNCTION VELO11_CUOR_EXECUTE.
**---------------------------------------------------------------------
*** Lokale Schnittstelle:
** IMPORTING
**   REFERENCE(DIALOGUE_ALLOWED_IV) TYPE C OPTIONAL
**   TABLES
**     VLCDIAVEHI_CT_STRUCTURE    VLCDIAVEHI
**     VLCH_MSSG_CT       VLCH_MSSG_PT
**     VLCBAPIOU_IT   TYPE     VLCBAPIOU_T
**   CHANGING
**   REFERENCE(VLCACTDATA_CS) TYPE     VLCACTDATA
**   EXCEPTIONS
**     DIFFERENT_KUNNR
**     SEVERAL_OFFERS_CONCERNED
**     CONTR_ENTITY_NOT_FOUND
**     DIFFERENT_CONTR_ENTITY
**     CUSTOMER_DATA_NOT_FOUND
**     VEHUSAGE_NOT_FOUND
**     PARTNER_ERROR
**     ACTION_NOT_PERFORMED
**     UPDATE_NOT_PERFORMED
**     SET_TARGET_CFG_ERROR
**---------------------------------------------------------------------
```

- **DIALOGUE_ALLOWED_IV** (optional)
  This is only required if screens are called in the EXECUTE module. This also applies to modules that are called by the EXECUTE module.

  This flag has a value of X if the action is called from transaction VELO; in this case the EXECUTE module may call screens.

  If the action is called by a VMS BAPI (batch job, IDoc interface, web interface), the flag is initial and screens may not be called in the EXECUTE module.

  Screens may only be called in the EXECUTE module if the variable DIALOGUE_ALLOWED_IV has a value of X. An appropriate IF statement must therefore be embedded in each screen.

- **VLCDIAVEHI_IT** (required)
  Internal table that contains all of the vehicles affected by the action. This table contains all the vehicle table fields and further fields that have been prepared, such as short texts and time stamps (date, time).

  Data from this internal table can be changed in the EXECUTE module. Further on in the program, this data is written to the vehicle table VLOVEHICLE and the vehicle history VLCHISTORY by means of MOVE-CORRESPONDING.
The configuration data can be transferred to the sales order or order BAPIs in the EXECUTE module in order to write the configuration to the standard documents, or you can use module VEO03_SET_SINGLE_CONFIG to store it as a vehicle configuration.

- **VLCH_MSSG_CT** (necessary)

  This is an internal table in which all the relevant messages (irrespective of type) that have occurred are collected. Above all, the messages have to be written to this table so that the user can be informed of the error by means of an error log.

  The EXECUTE module calls module VEO03_FILL_ERROR_TABLE in order to fill the table (see: 3.5.4.2.1).

- **VLCACTDATA_CS** (necessary)

  This is a communication structure that supports the data flow from the PREPARE module via the action screen to the EXECUTE module and from there back to the action screen. Additional data for the execution of the action is read from this structure.

  Furthermore, data that is to be displayed on the action screen can be written to this structure. Example: if a document has been changes in the EXECUTE module, the document number can be written to this structure so that it is displayed on the action screen.

Because of the existence of interlinking actions, an EXECUTE module may NEVER delete the content of this structure, it should only ever add to it.

- **Exceptions** (mostly necessary)

- **Exceptions** (mostly necessary)

  There is a basic rule in VMS programming: error messages should not be triggered within a function module; they should rather be handed over step by step to the interface.

  In the EXECUTE modules, messages have to be written to message table VLCH_MSSG_CT so that they can be made available to the user in the form of an error log.

  If an error situation occurs in the EXECUTE module, the error message is written to message table VLCH_MSSG_CT and the error situation is transferred to the calling program by means of a RAISE statement (coding example: RAISE VEHIUSAGE_NOT_FOUND).

  If the EXECUTE module triggers an error, the action is terminated for all the vehicles that were transferred to the module.
3.5.4 Source Text

3.5.4.1 Data Declarations

```
* Action CUOR: Creates the Sales Order
* All error messages are written into table VLCH_MSSG_CT

* Declarations
* ===========

* Local structure for data of VLCDIAVEHI_CT
  DATA: VLCDIAVEHI_LS TYPE VLCDIAVEHI,
* Action specific header data of VLNACTDATA_CS
  VLCACTDATA_HEAD_LS TYPE VLCACTDATA_HEAD_S,
* Action specific Item data of VLCACTDATA_CS
  VLCACTDATA_ITEM_LS TYPE VLCACTDATA_ITEM_S,
  VLCACTDATA_ITEM_LT TYPE VLCACTDATA_ITEM_T,
* Local structure / table for the Sales Order table of the VMS
  VLCCUORDER_LS LIKE VLCCUORDER,
  VLCCUORDER_LT LIKE TABLE OF VLCCUORDER,
* List of vehicles relevant for the action
  VLCGUID_LT LIKE TABLE OF VLCGUID,
  VLCGUID_LS LIKE VLCGUID,
* Structure and table for action document type
  ACTDOCTYPE_LT TYPE VLCH_ACTDOCTYPE_PT,
  ACTDOCTYPE_LS TYPE VLCH_ACTDOCTYPE_PS,
* Configuration stuff
  VLBAPICU_LS TYPE VLBAPICU,
* Local structure / table for BAPI_SALESORDER_CREATEFROMDAT2
  ...
* Other variables
  ...
```

- **VLCDIAVEHI_LS**
  Local vehicle structure that is required when vehicle table VLCDIAVEHI_IT (interface, see: 3.5.3) has to be read or changed.

- **VLCACTDATA**
  Data objects that are required when you work with header or item data (see: 1.5) from the communication structure.

- **VLCCUORDER_LT**
  VMS document table for the standard document Sales Order. There are separate VMS tables for the different document categories (see: 4.1.1).
  
  The relationship between vehicle and document number/item is stored in the document tables. If you know the vehicle, you can determine the appropriate documents in these tables.

© 2003 SAP AG
Neuortstr. 16
D-69190 Walldorf

Title: Vehicle Management System: Creating Actions
Version: 5.0
Date: 26.03.2003

Page 59 of 78
• **VLCGUID_LT**  
  GUID (globally unique identifier), in other words the key from vehicle table VLCVEHICLE that uniquely identifies a vehicle.  
  Here an internal table is declared that can be filled with the GUIDs of the appropriate vehicles in order to select data from different tables.

• **ORDER_ACTDOCTYPE_LT**  
  Internal table that is filled with the action document category in order to select data from the appropriate VMS document tables (see: 3.5.4.2.1).

• **VLCBAPICU_LS**  
  Local configuration data structure that is required when internal table VLCBAPICU_IT (interface, see: 3.5.3) has to be read.

• Local structure/table for BAPIs  
  The interface parameters must have been filled before a standard BAPI can be called in the EXECUTE module. The data objects required are declared here.

• Other variables  
  It is mostly simple and unstructured variables that are declared here.

### 3.5.4.2 Checks

All the checks implemented in the PREPARE modules have to be repeated in the EXECUTE modules.

To do this, you can either implement these checks directly in the EXECUTE module or call the PREPARE module in the EXECUTE module. You should decide from case to case which of these methods is the most suitable.

#### 3.5.4.2.1 Reading and Checking Preceding Document

This is a simplified extract from the coding for the EXECUTE module of action DECO *Delete Sales Order*: If you want to be able to delete a sales order, a sales order must already exist for the appropriate vehicles.
* Step 1
* Fill table of GUIDs (needed as import parameter for function module
* VEO04_READ_SALESORDER)

LOOP AT VLCDIAVEHI_CT INTO VLCDIAVEHI_LS.

CLEAR: VLCGUID_LS.

VLCGUID_LS-VGUID = VLCDIAVEHI_LS-VGUID.
APPEND VLCGUID_LS TO VLCGUID_LT.

ENDLOOP.

* Step 2
* Read the entries for the selected vehicles from the Sales Order table
* of the Vehicle Manager

* Action CUOR
ACTDOCTYPE_LS-SIGN = 'I'.
ACTDOCTYPE_LS-OPTION = 'EQ'.
ACTDOCTYPE_LS-LOW = ABA_CUOR_GC.
APPEND ACTDOCTYPE_LS TO ACTDOCTYPE_LT.

* Step 3
* Read the sales orders from the vehicle manager table vlcuorder
CALL FUNCTION 'VEO04_READ_SALESORDERS'
EXPORTING
  LATEST_IV = 'Y' " flag added for CRT4
TABLES
  VLCGUID_LT = VLCGUID_LT
  VLACTIONTYPE_LT = ACTDOCTYPE_LT
  VLCUORDER_LT = VLCCUORDER_LT
EXCEPTIONS
  NO_DATA_RECEIVED = 1
  NOTHING_FOUND = 2
  OTHERS = 3.

* Error handling
IF SY-SUBRC NE 0.

CALL FUNCTION 'VEO03_FILL_ERROR_TABLE'
EXPORTING
  MSGID_IV = SY-MSGID
  MSGTY_IV = SY-MSGTY
  MSGNO_IV = SY-MSGNO
  MSGV1_IV = SY-MSGV1
  MSGV2_IV = SY-MSGV2
  MSGV3_IV = SY-MSGV3
  MSGV4_IV = SY-MSGV4
TABLES
  VLMSSG_CT = VLMSSG_CT.
RAISE NO_SALES_ORDER_FOUND.

ENDIF.

*---------------------------------------------------------------
• Step 1
  Internal table VLCGUID_LT is filled with the GUIDs of the appropriate vehicles.

• Step 2
  Internal table ACTDOCTYPE_LT is filled with the action document category.

• Step 3
  All the sales orders (document number/item) for the vehicles affected are read in VMS document table VLCCUORDER.
  You can find further information on action document categories under 3.4.4.2.1.
3.5.4.2.2 Calling the PREPARE Module

So that you do not have to implement checks more than once, you can call the PREPARE module in the EXECUTE module. To do this, set import parameter XCALLEDBYEXECUTE_IV (see:3.4.3) to the value X so that you can react logically to this situation in the PREPARE module. The PREPARE module, for example, should not usually prepare any more data when the EXECUTE is called.

All the checks that are switched off for interlinking actions when the PREPARE is called (for example, check for the existence of preceding documents, see: 3.4.4.2.2) have to be run through when the EXECUTE is called.

This is a simplified extract from the coding for action MORD Change Purchase Order. The EXECUTE module calls the PREPARE module.

```
*=================================================================================================
* # Step 1 #
* PREPARATIONS
*=================================================================================================

* Read necessary data, perform some checks. Most of this is also done
* in VEL010_MORD_PREPARE. In case of interlinked actions and action
* execution without subscreens (IDOC, BAPI) it is necessary to perform
* this in the VEL010_MORD_EXECUTE function module, too

* Perform preparations (again)
CALL FUNCTION 'VEL010_MORD_PREPARE'
  EXPORTING
    XINTERLINKED_IV = X
    XCALLEDBYEXECUTE_IV = 'X'
  TABLES
    VLCDIAVEHI_IT = VLCDIAVEHI_CT
    VLECADATA_CS = VLECADATA_CS
EXCEPTIONS
    VEHICLES_OF_SEVERAL_ORDERS = 1
    NO_ORDERS_FOUND_IN_VLC = 2
    ERROR_IN_BAPI_PO_GETDETAIL = 3
    NOT_PERFORMED = 4
    OTHERS = 5.

* Error handling
IF SY-SUBRC <> 0.
  CALL FUNCTION 'VEL003_FILL_ERROR_TABLE'
  EXPORTING
    MSGID_IV = SY-MSGID
    MSGTY_IV = SY-MSGTY
    MSGNO_IV = SY-MSGNO
    MSGV1_IV = SY-MSGV1
    MSGV2_IV = SY-MSGV2
    MSGV3_IV = SY-MSGV3
    MSGV4_IV = SY-MSGV4
    VLCH_MSSG_CT = VLCH_MSSG_CT.
  RAISE PREPARE_ERROR.
ENDIF.  ----------------------------------------------
```

© 2003 SAP AG  Title: Vehicle Management System: Creating Actions
Neurottstr. 16  Version: 5.0
D-69190 Walldorf  Date: 26.03.2003
If you call the PREPARE module in the EXECUTE module, parameter XCALLEDBYEXECUTE_IV is set to value X. Parameter XINTERLINKED_IV cannot be filled and is therefore always initial.

3.5.4.3 Creating/Changing/Deleting SAP Standard Documents

You should, as far as possible, only process SAP standard documents with BAPIs. You have to execute three main steps in the coding:

1. Read data
2. Convert/copy data to the BAPI structures
3. Call BAPI

The following simplified coding example shows the calling of a BAPI for creating a sales order in the EXECUTE module for action CUOR.

```
* Step B.2
* Create the customer order
*
* Call the function module SD_SADESDOCUMENT_CREATE
CALL FUNCTION 'BAPI_SALESORDER_CREATEFROMDAT2'
  EXPORTING
    SALESDOCUMENTIN = SALESDOCUMENTIN_LV
    ORDER_HEADER_IN = SD_HEADER_LS
    ORDER_HEADER_INX = ORDER_HEADER_INX_LS
    SENDER = SENDER_LV
    BINARY_RELATIONSHIP_TYPE = BINARY_RELATIONSHIP_TYPE_LV
    INT_NUMBER_ASSIGNMENT = INT_NUMBER_ASSIGNMENT_LV
    BEHAVE_WHEN_ERROR = BEHAVE_WHEN_ERROR_LV
    LOGIC_SWITCH = LOGIC_SWITCH_LS
    TESTRUN = TESTRUN_LV
    CONVERT = CONVERT_LV

IMPORTING
    SALESDOCUMENT = VBELN_LV

TABLES
    RETURN = RETURN_LT
    ORDER_ITEMS_IN = SD_ITEM_LT
    ORDER_ITEMS_INX = ORDER_ITEMS_INX_LT
    ORDER_PARTNERS = SD_PART_LT
    ORDER_SCHEDULES_IN = SD_SCHE_LT
    ORDER_SCHEDULES_INX = ORDER_SCHEDULES_INX_LT
    ORDER_CONDITIONS_IN = ORDER_CONDITIONS_IN_LT
    ORDER_CONDITIONS_INX = ORDER_CONDITIONS_INX_LT
    ORDER_CONFIGS_REF = BAPICUCFG_LT
    ORDER_CONFIGS_INST = BAPICUISLS_LT
    ORDER_CONFIGS_PART_OF = BAPICUPRT_LT
    ORDER_CONFIGS_VALUE = BAPICUVAL_LT
    ORDER_CONFIGS_BLOB = ORDER_CONFIGS_BLOB_LT
    ORDER_CONFIGS_VK = BAPICUVK_LT
    ORDER_CONFIGS_REFINST = BAPICUREF_LT
    ORDER_CCARD = ORDER_CCARD_LT
    ORDER_TEXT = ORDER_TEXT_LT
    ORDER_KEYS = ORDER_KEYS_LT
```
* Step B.3
* Error handling for the BAPI
* Clear the error-flag
  CLEAR RAISE_FLAG_LV.
* Loop over the return-message-table and write ALL messages into the
  Vehicle Manager message table VLCH_MSSG_CT
  LOOP AT RETURN_LT INTO RETURN_LS.

  CALL FUNCTION 'VELO03_FILL_ERROR_TABLE'
    EXPORTING
      MSGID_IV = RETURN_LS-ID
      MSGTY_IV = RETURN_LS-TYPE
      MSGNO_IV = RETURN_LS-NUMBER
      MSGV1_IV = RETURN_LS-MESSAGE_V1
      MSGV2_IV = RETURN_LS-MESSAGE_V2
      MSGV3_IV = RETURN_LS-MESSAGE_V3
      MSGV4_IV = RETURN_LS-MESSAGE_V4
    TABLES
      VLCH_MSSG_CT = VLCH_MSSG_CT.
  * Set the error flag if there are messages with a specific
    message-type
    IF RETURN_LS-TYPE = 'E' OR RETURN_LS-TYPE = 'A'.
      RAISE_FLAG_LV = XFLAG_GC.
    ENDF.

  ENDOLOOP. "RETURN_LT
* If the error-flag is set, raise an error !
* IF NOT RAISE_FLAG_LV IS INITIAL.
  RAISE ACTION_NOT_PERFORMED.
  ENDF.

Many BAPIs return a RETURN table in which all the messages that have been collected are stored. During the loop processing via this table, every message is written to message table VLCH_MSSG_CT by using module VELO03_FILL_ERROR_TABLE; the messages are then available in the action log.

If an error message or a cancellation/termination message occurs, error indicator RAISE_FLAG_LV is set and then the error situation is transferred to the calling program by using the RAISE statement.

Customer-specific fields can be transferred to many BAPIs by using data object EXTENSIONIN.
3.5.4.4 Updating VMS Document Tables

The relationship between vehicle and document number/item is stored in the VMS document tables. Therefore an entry must be added to or deleted from the appropriate VMS document table (see: 1.4) if a document has been created or deleted.

The following simplified coding example shows the appropriate parts of coding in the EXECUTE module for action CUOR *Create Sales Order*.

```
* Step 1
* Write entries to the sales order table of the vehicle

REFRESH VLCCUORDER_LT[].
LOOP AT VLCDIAVEHI_CT INTO VLCDIAVEHI_LS.
    CLEAR: VLCCUORDER_LS.
    LOOP_INDEX_LV = SY-TABIX.

* Sales order table

READ TABLE SD_ITEM_LT INTO SD_ITEM_LS INDEX LOOP_INDEX_LV.

VLCCUORDER_LS-VGUID = VLCDIAVEHI_LS-VGUID.
VLCCUORDER_LS-TSTMP = VLCDIAVEHI_LS-NEWTSP.
VLCCUORDER_LS-ACTDOCTYPE = ABA_CUOR_GC.
VLCCUORDER_LS-VBELN = VBEIN_LV.
VLCCUORDER_LS-POSNR = SD_ITEM_LS-ITEM_NUMBER.
VLCCUORDER_LS-BSTNK = VLCACTDATA_HEAD_LS-BSTNK.
VLCCUORDER_LS-ERNAM = VLCDIAVEHI_LS-ACTERNAM.

APPEND VLI CUORDER_LS TO VLCCUORDER_LT.

ENDLOOP.  " VLCDIAVEHI_CT

* Step 2
* Write an entry in the table linking sales doc with vehicle

CALL FUNCTION 'VELO04_UPDATE_VLCCUORDER' IN UPDATE TASK

TABLES
   VLCCUORDER_LT = VLCCUORDER_LT

EXCEPTIONS
   NO_DATA_RECEIVED = 1
   NO_UPDATE_PERFORMED = 2
   OTHERS = 3.

* # Begin error handling
IF SY-SUBRC <> 0.
   CALL FUNCTION 'VELO03_FILL_ERROR_TABLE'
   EXPORTING
      MSGID_IV = SY-MSGID
      MSGTY_IV = SY-MSGTY
```

© 2003 SAP AG  Title: Vehicle Management System: Creating Actions
Neurollstr. 16  Version: 5.0
D-69190 Walldorf  Date: 26.03.2003
The internal table (here: VLCCUORDER_LT) is now transferred to the update module, there will be data inconsistencies.

Database updates are basically encapsulated in separate update modules in VMS.

3.5.4.5 Update

- **LUW (Logical Unit Of Work)**
  The control over the LUW has to lie with module Velo09_SET_ACTION.

  - The action-specific EXECUTE modules should not use a COMMIT. If they do, there will be data inconsistencies.

  In other words: SAP standard modules that use a COMMIT may not be called in the EXECUTE module. Furthermore, the SUBMIT <REPORT> command is not allowed, since this automatically causes a COMMIT. Just as unsuitable is the use of the command CALL TRANSACTION since the control is relayed to another transaction.

- **Use update modules in the EXECUTE module**

  Database updates are basically encapsulated in separate update modules in VMS. This must be set in the module attributes (update module; start immediately).
The update modules are called with addition IN UPDATE TASK (example: 3.5.4.4). This means that the module is only run through when the COMMIT command is used later. If you want to debug these modules, you have to make the setting for update debugging in the ABAP Debugger.

- Synchronous versus asynchronous (see: 1.3.2.2):

Some actions are updated asynchronously; in other words, if the user has already received the message in transaction VELO that the action has been successfully executed, the update task is possibly still busy working.

For interlinking actions, the individual elementary actions are executed one after the other and updated separately and synchronously. Not until the preceding elementary action has been updated is the following elementary action started.

If you want to implement a different behaviour, you can implement the COMMIT method of BAAdI VLC_EXECUTE_ACTION.

### 3.5.4.6 Writing Vehicle Data

Data has to be written to internal table VLCDIAVEHI_CT in the EXECUTE module if it is to be written to the vehicle table or the history table.

The following simplified coding example shows the appropriate parts of coding in the EXECUTE module for action CUOR *Create Sales Order*.

```
*-----------------------------------------------
* Step C.2
* Write entries to the sales order table of the vehicle manager and to
* the vehicle table VLCDIAVEHI_CT

  LOOP AT VLCDIAVEHI_CT INTO VLCDIAVEHI_LS.

  * Find the correct line of VLCACDATA
    READ TABLE VLCACDATA_ITEM_LT INTO VLCACDATA_ITEM_LS
    WITH KEY VGUID = VLCDIAVEHI_LS-VGUID.

  * Vehicle data
    VLCDIAVEHI_LS-KUNNR  = VLCACDATA_HEAD_LS-KUNNR.
    VLCDIAVEHI_LS-CU_NAME1 = VLCACDATA_HEAD_LS-KUNAM.
    VLCDIAVEHI_LS-ENDCU  = VLCACDATA_HEAD_LS-ENDCU.
    VLCDIAVEHI_LS-VHUSG  = VLCACDATA_HEAD_LS-VHUSG.
    VLCDIAVEHI_LS-VHUSGTXT = VHUSGT_LV.
    VLCDIAVEHI_LS-ACTDOCTYPE = ABA_CUOR_GC.

    MODIFY VLCDIAVEHI_CT FROM VLCDIAVEHI_LS INDEX LOOP_INDEX_LV.

  -----

  ENDL Ghost.  " VLCDIAVEHI_CT
```

© 2003 SAP AG  |  Title: Vehicle Management System: Creating Actions
Neurottstr. 16  |  Version: 5.0
D-69190 Walldorf |  Date: 26.03.2003
The data is written to internal table VLCDIAVEHI_CT during loop processing across the vehicle table. The new data is usually transferred from communication structure VCLACTDATA. Data from the header structure VCLACTDATA_HEAD_LS is valid for all the vehicles, data from the item structure VCLACTDATA_ITEM_LS is only ever valid for the current vehicle.

If a document was created in the action, the action document category must be set. This means, amongst other things, that a document icon is displayed in the vehicle history in transaction VELO.

3.5.4.7 Filling the Communication Structure

The EXECUTE module should, on no account, delete VCLACTDATA_CS, otherwise you cannot interlink actions. You may, however, add data.

3.5.4.7.1 Header Data

The document number of the newly created order, for example, is written to the header segment of the communication structure in the EXECUTE module for action CUOR *Create Sales Order*. This allows the document number to be displayed on the action screen after the action has been executed.

```plaintext
                        *******************************************
                        | processing block C                          |
                        |                                            |
                        | Prepare the return data                      |
                        *******************************************

* Step C.0
* Fill the document number into VCLACTDATA_HEAD_S for displaying on
* dynpro
  VCLACTDATA_CS-VBELN = VBELN_LV.
```
### 3.5.4.7.2 Item Data

- PREPARE table containing existing old data of VLCACTDATA
  VLCACTDATA_ITEM BEFORE_LT[] = VLCACTDATA_CS-ACTDATA_ITEM.

- One line per vehicle ...
  LOOP AT VLCDIAVEHI_CT INTO VLCDIAVEHI_LS.

- Don't mix data of different vehicles
  CLEAR VLCACTDATA_ITEM_LS.

- Use existing old data of VLCACTDATA_CS (This read does only find
  entries in case of interlinked actions)
  READ TABLE VLCACTDATA_ITEM BEFORE_LT INTO VLCACTDATA Item_LS
    WITH KEY VGUID = VLCDIAVEHI_LS-VGUID.

- Write GUID into the new VLCACTDATA (needed if VLCACTDATA ITEM_LS is
  initial)
  VLCACTDATA_ITEM_LS-VGUID = VLCDIAVEHI_LS-VGUID.

- Add action-specific data to VLCACTDATA ITEM_LS

- Add line to new VLCACTDATA
  APPEND VLCACTDATA ITEM_LS TO VLCACTDATA ITEM AFTER_LT.

ENDLOOP.

- Replace old VLCACTDATA by new VLCACTDATA. Contains the entries of the
  old VLCACTDATA and in addition data of the action.
  CLEAR VLCACTDATA_CS-ACTDATA ITEM.
  VLCACTDATA CS-ACTDATA ITEM[] = VLCACTDATA ITEM AFTER_LT[].

- First of all, field VLCACTDATA-CS-ACTDATA ITEM (item data) is written to the BEFORE table.
- If it exists, the line that fits the current vehicle is read from this old VLCACTDATA version.
  The action-specific data (in particular the GUID) is added to this line and then appended to the
  AFTER table.
- Finally, field VLCACTDATA_CS-ACTDATA ITEM is deleted and overwritten by the AFTER table.

This procedure ensures that the table is not updated but merely enhanced with additional new
data. This is of particular importance when you are dealing with the interlinking of actions.
3.5.5 Call

An implementation is created for BAdI VLC_EXECUTE_ACTION in transaction SE18. The action-specific EXECUTE module is called in BAdI method EXECUTE_FURTHER_ACTIONS. Example of coding:

```
CASE ACTION_TO_BE_PERFORMED_IV.

WHEN '<Aktionskürzel1>'. " Customer's first action !
CALL_FUNCTION 'FugrName>_Aktionskürzel1>_EXECUTE'
EXPORTING
  DIALOGUE_ALLOWED_IV = DIALOGUE_ALLOWED_IV
TABLES
  VLCDIAVEHI_CT = VLCDIAVEHI_CT
  VLCH_MSSG_CT = VLCH_MSSG_CT
  VLCBAPICU_IT = VLCBAPICU_IT
CHANGING
  VLCACTDATA_CS = VLCACTDATA_CS
EXCEPTIONS
  ACTION_NOT_PERFORMED = 1
  UPDATE_NOT_PERFORMED = 2
  OTHERS = 3.

IF SY-SUBRC <> 0.
  THE error message(s) are written into VLCH_MSSG_CT by the
  function module
  RAISE ACTION_NOT_PERFORMED.
ENDIF.

WHEN '<Aktionskürzel2>'. " Customer's second action !

... usw.

ENDCASE.
```
3.6 Branching from Vehicle History to Document

When the system executes an action, an entry is automatically made in the history table.

If the action created a document, this is displayed in the history entry by a special document icon. The relationship between vehicle and document number is stored in the appropriate VMS document table. If a user chooses the document icon, the document is determined from the VMS table and automatically displayed in the appropriate transaction.

This behavior is controlled in function module VELO04_READ_TRANSCALLDATA. The system controls whether an appropriate standard document is displayed or whether a message appears informing the user that there is no standard document for the chosen history entry in accordance with the four-character action key.

Function module VELO04_READ_TRANSCALLDATA calls a BAdI in which you can implement customer enhancements; this is particularly necessary if you have created a customer-specific VMS document table.

Branching from an entry in the history table to a transaction for displaying the document is implemented in BAdI VLC_HISTORY_CALL_TRA.

The document icon is then automatically displayed when field ACTDOCTYPE of table VLCDIAVEHI_CT has been filled in the EXECUTE module of the action that created the document.
4 Appendix

4.1 Tables in VMS

4.1.1 Transaction Data

<table>
<thead>
<tr>
<th>Transaction Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLCVEHICLE</td>
<td>VELO : Vehicle</td>
</tr>
<tr>
<td>VLCADDDATA</td>
<td>VELO : Qualifier Data for Vehicle</td>
</tr>
<tr>
<td>VLCADDCFG</td>
<td>VELO : Additional Vehicle Configuration (Not Relevant to Search)</td>
</tr>
<tr>
<td>VLCHISTORY</td>
<td>VELO : Vehicle History</td>
</tr>
<tr>
<td>VLCCUORDER</td>
<td>VELO : Sales Order</td>
</tr>
<tr>
<td>VLCDELIVERY</td>
<td>VELO : Delivery</td>
</tr>
<tr>
<td>VLCGISSUE</td>
<td>VELO : Material Document / Goods Issue</td>
</tr>
<tr>
<td>VLCGMOVEM</td>
<td>VELO : Material Documents (Other)</td>
</tr>
<tr>
<td>VLCGRECEIPT</td>
<td>VELO : Material Document / Goods Receipt</td>
</tr>
<tr>
<td>VLCINCINVOICE</td>
<td>VELO : Incoming Invoice</td>
</tr>
<tr>
<td>VLCINQUIRY</td>
<td>VELO : Inquiry/Request for Quotation</td>
</tr>
<tr>
<td>VLCINVOICE</td>
<td>VELO : Billing Document</td>
</tr>
<tr>
<td>VLOFFER</td>
<td>VELO : Quotation</td>
</tr>
<tr>
<td>VLCPOORDER</td>
<td>VELO : MM Order</td>
</tr>
<tr>
<td>VLCRCSTRORD</td>
<td>VELO : Subcontract Order / Rework</td>
</tr>
<tr>
<td>VLCRESERVATION</td>
<td>VELO : Reservation</td>
</tr>
<tr>
<td>VLCDEAEND</td>
<td>VELO : Relationship Between Dealer and End Customer</td>
</tr>
<tr>
<td>VLCIDGU</td>
<td>VELO : Vehicle to IDoc Number Assignment Table</td>
</tr>
<tr>
<td>VLC_DBCONSTATUS</td>
<td>VELO : Status of LiveCache Connection (Obsolete)</td>
</tr>
<tr>
<td>CVLC_TSTMP</td>
<td>VELO : Time of Last BW Extraction</td>
</tr>
</tbody>
</table>
4.1.2 Master Data

<table>
<thead>
<tr>
<th>Master Data Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVLC05</td>
<td>VELO: Control Table Object Characteristics</td>
</tr>
<tr>
<td>CVLC20</td>
<td>VELO: Filter Definition IBase =&gt; LiveCache (Obsolete)</td>
</tr>
<tr>
<td>CVLC23</td>
<td>VELO: Assign Organization Data to VMS Roles</td>
</tr>
<tr>
<td>CVLC24</td>
<td>VELO: Assign Vehicle Model to VMS Roles</td>
</tr>
<tr>
<td>CVLC25</td>
<td>VELO: Assign User to VMS Roles</td>
</tr>
<tr>
<td>VLCCHPROFIND</td>
<td>VELO: Configuration Change Profile Determination Profile</td>
</tr>
<tr>
<td>VLCCHPROHEAD</td>
<td>VELO: Configuration Change Profiles</td>
</tr>
<tr>
<td>VLCCHPROHEADT</td>
<td>VELO: Text for VLCCHPROHEAD</td>
</tr>
<tr>
<td>VLCCHPROITEM</td>
<td>VELO: Configuration Change Profiles: Characteristic Assignment</td>
</tr>
<tr>
<td>VLCFINDCNTRL</td>
<td>VELO: Action Control Determination</td>
</tr>
<tr>
<td>VLCSCAMPGN</td>
<td>VELO: Sales Campaign</td>
</tr>
<tr>
<td>VLCSCAMPGNT</td>
<td>VELO: Text for VLCSCAMPGN</td>
</tr>
<tr>
<td>CVLC_CHAR_LINK</td>
<td>VELO: Assignment of Class Characteristics to BW Characteristics</td>
</tr>
<tr>
<td>VLCSEARCHPROF</td>
<td>VELO: Profile for Vehicle Search</td>
</tr>
<tr>
<td>VLCSEARCHPROFT</td>
<td>VELO: Text Table: Profile for Vehicle Search</td>
</tr>
<tr>
<td>VLCSEARCHFIELD</td>
<td>VELO: Search Fields (Selections) Vehicle Search (for Profile)</td>
</tr>
<tr>
<td>VLCSEARCHCHAR</td>
<td>VELO: Search Fields (Characteristics) Vehicle Search (for Profile)</td>
</tr>
<tr>
<td>VLCBATCHACT</td>
<td>VELO : Variant for Action Execution in Background</td>
</tr>
</tbody>
</table>

4.1.3 Customizing Tables

<table>
<thead>
<tr>
<th>Customizing Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVLC01</td>
<td>VELO : Action Control</td>
</tr>
</tbody>
</table>

© 2003 SAP AG
Neurottstr. 16
D-69190 Walldorf

Title: Vehicle Management System: Creating Actions
Version: 5.0
Date: 26.03.2003
CVLC01T    VEL0: Text for CVLC01
CVLC02    VEL0: Vehicle Status
CVLC02T    VEL0: Text for CVLC02
CVLC03    VEL0: Actions
CVLC03T    VEL0: Text for CVLC03
CVLC03I    VEL0: Interlinking Actions
CVLC04    VEL0: Action Matrix
CVLC06    VEL0: Current Location
CVLC06T    VEL0: Text for CVLC06
CVLC11    VEL0: Availability, Locations, Planned Delivery Time
CVLC12    VEL0: Status - Availability, Locations, Planned Delivery Time
CVLC10    VEL0: Availability Status
CVLC10T    VEL0: Text for CVLC10
CVLC13    VEL0: Vehicle Usage
CVLC13T    VEL0: Text for CVLC13
CVLC14    VEL0: Action Determination via External Status
CVLC15    VEL0: External Status
CVLC15T    VEL0: Text for CVLC15
CVLC16    VEL0: Assignment of Qualifiers to Actions
CVLC17    VEL0: Assignment of Vehicle Attributes to Actions
CVLC18    VEL0: Vehicle Search Area
CVLC18T    VEL0: Text for CVLC18
CVLC18V    VEL0: Vehicle Search Area – Sales Organization
CVLC18W    VEL0: Vehicle Search Area - Plant
CVLC19    VEL0: Define VMS Roles
CVLC19T    VEL0: Text for CVLC19
CVLC21    VEL0: Assign Categories
CVLC22    VEL0: Assign Message Determination Procedure to Plant
CVLC26    VEL0: Assign Own Dialog Messages
CVLC27    VEL0: Access Authorization During Vehicle Search
CVLC27T    VEL0: Text for CVLC27
CVLCQUA    VEL0: Define Qualifiers
CVLCQUAT    VEL0: Text for CVLCQUA
4.1.4 Control Tables

<table>
<thead>
<tr>
<th>Control Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLCSEARCHCONTROL</td>
<td>VEO: Control Table Search Function</td>
</tr>
<tr>
<td>VLCGUIDETABLES</td>
<td>VEO: Listing of GUID-VMS Tables Relevant to Search</td>
</tr>
<tr>
<td>VLCVALIDFIELDS</td>
<td>VEO: Valid Vehicle Attributes from VLCDIAVEHI Structure</td>
</tr>
</tbody>
</table>

4.2 Guidelines for Programming in VMS

The following guidelines were used as far as possible when VMS and its actions were developed. This simplifies the searches, reading and understanding of the individual objects and coding. You can build on these guidelines for customer-specific developments.

Object Definitions:

- Development language: German
- Development abbreviation: VLC
- Development class/package: ISAUTO_VLC
- Database tables: 6-character, beginning with VLCxxx, appropriate text tables VLCxxxT
- Customizing tables: 7-character, beginning with CVLCxxx, appropriate text tables CVLCxxxT
- Structures: 8-character, beginning with VLCxxxxx
- Views: 6- or 8-character, beginning with V_VLCxxx
- Customizing views: V_CVLcxx
- View clusters: VC_VLCxxx
- Data elements: beginning with VLC.....
- Domains: beginning with VLC.....
- Lock objects: E_VLCxxx
- Function groups: VELOXX
- Function modules: VELOXX_<OBJECT>_<FUNCTION>_<EXTENSION>
• Reports: 10-character, RVLCxxxx
• Module pools: 10-character, SAPMJITxxx
• Message class: 5-character, VELOx
• Transaction codes: 5-character, beginning with VELOx
• Customizing Trx: 6-, beginning with OVLcxx
• SET/GET parameters: beginning with VELO_
• FORM routines: beginning with VLC_
• Includes: beginning with LVLC.....
• Includes/constraints: beginning with LVLCXXCON

**Variable Definitions**
The ABAP Style Guide was used when the variables were defined:

• Variable names
  - Name + administration type + category

• Administration type
  - Import (i)
  - Export (e)
  - Changing (c)
  - Tables (t)
  - Local (l)
  - Global (g)
  - Statics (s)
  - Parameter (p)

• Category
  - Variable (v)
  - Table (t)
  - Structure (s)
  - Constant (c)
  - Field (f)

• Example
  - number_iv, counter_lv, jtcu_gt, ...
General Guidelines:

- Always declare data dictionary tables (TABLES statement) globally. It is best to do without the TABLES statement and only use them where absolutely necessary (for example screens, see: global variables).
- Instead of `data vlcvehicle_Lt like jil occurs 0 with header line, it is better to use: data vlcvehicle_Lt like vlcvehicle occurs 0 and then, in addition: data vlcvehicle_Ls like vlcvehicle. In other words, do not use ltabs with a header line but rather wa & ltab without header line).
- Keep the number of global variables to a minimum – only use global variables if they are absolutely necessary and it makes sense to do so (data buffers, ALV, screen etc.).
- Avoid hidden transfer (global variable).
- External FORM calls are not allowed (possible within function group, otherwise via function module).
- All names (including table field names and structures) should be in English and please avoid using misleading names at all costs.
- Please write commentaries in English and document clearly and well. If need be, you can comment in German and English to avoid any misunderstandings.
- Use search helps (DDIC) instead of help views, matchcodes etc.
- A maximum of one statement per line (helpful when you are using the debugger and it is also clearer like this).
- Place parts of the program that can be used generally in function modules (however, do not atomize).
- Avoid coding for which the commentary is blanked out or non-existent.
- Isolate subtasks that can be isolated and program them so that they can be re-used.
- Define all screen fields in the data dictionary (DDIC).
- Define global work fields in the data dictionary (DDIC).
- Use a separate include for each FORM routine (maximum of 500 lines of coding is better).
- Raise every error message. Do not use RAISE, but rather MESSAGE .... RAISING.