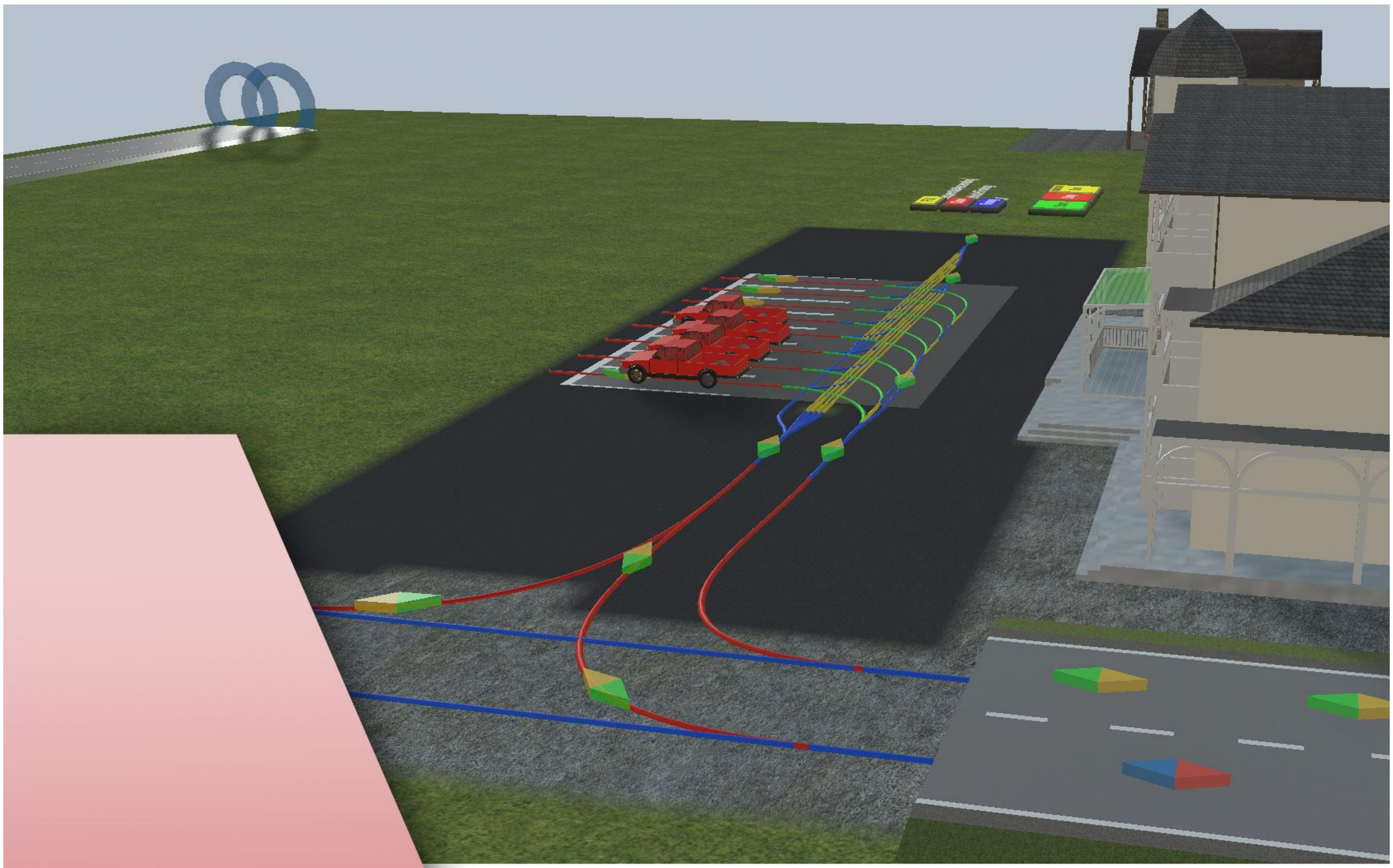
MBS Companion



VP

Lua Module

Vehicle Parking

VP – Feature Summary (not all are implemented yet)

"Vehicle Parking" manages structured parking areas using virtual tracks and contacts.
It controls how vehicles **enter**, **wait**, **queue**, and **leave** a parking zone, and ensures orderly flow even with many vehicles.

Core Features

- **Parking Zone Management**
  VP defines one or more virtual parking areas composed of parking lanes, holding tracks, or single park slots. The module keeps track of vehicles currently parked and their order.

- **Entry Control**
  VP manages how vehicles are admitted:
  – allow entry only if capacity is available
  – single-lane entry – maybe later; multi-lane entry
  – optional "entry queue" before the parking area using SS module

- **Parking Order & Slot Assignment**
  Depending on configuration, VP supports:
  – first-free-slot assignment
  – sequential or random slot filling
  – simple queue-based storage

- **Timed or Condition-Based Release**
  Vehicles can leave the parking area based on:
  – timers (e.g., fixed parking duration)
  – external triggers
  – vehicle specific timing
  – downstream modules requesting a vehicle (e.g., FD, VO, SS)

- **Vehicle Tracking**
  VP stores metadata for each parked vehicle, such as:
  – time of entry (for duration-based logic)
  – assigned slot/lane
  – return destination or forward target
  – optional user-defined variables

- **Flow Control Within the Parking Area**
  VP can coordinate internal movements, such as:
  – advancing a queue when a slot becomes free
  – reversing out of slot and progressing to exit lane
  – repositioning vehicles from a holding lane to a free parking slot

- **Basic Error Handling & Safety**
  – detects full parking zones and blocks new entries
  – avoids collisions by sequential slot movement
  – ensures vehicles are not lost or stuck without a valid next target

- **Fill Level Indicator**
  – fill level can be queried
  – overflow area can be linked

VW – Feature Summary
"Vehicle Wait" manages controlled queues for temporary vehicle stoppage.
It is used for bus lanes, taxi queues, drop-off/pick-up points, service lanes, and other situations where vehicles must pause before continuing.

Core Features

- **Queue Formation & Ordering**
  VW creates orderly queues using virtual tracks or dedicated waiting lanes.
  Vehicles join the queue in sequence, ensuring clean and predictable flow.

- **Controlled Release**
  Vehicles are released one at a time based on:
  – fixed or variable timers
  – downstream availability
  – external triggers (e.g., passengers boarding)

- **Multi-Vehicle Waiting Behavior**
  Supports single-lane or multi-lane queues, with optional merging back into main traffic flows.

- **Scheduling Integration**
  VW can serve as a timed staging point before vehicles enter route modules.
  This enables simple scheduling patterns such as:
  – service intervals (e.g., buses leaving every X minutes)
  – staggered departures
  – scheduled feeder traffic into *SR* or *SS* modules

- **Configurable Stop Logic**
  – minimum wait time
  – maximum queue length
  – optional bypass logic for through-traffic (when combined with FD)

- **Cooperation with Other Modules**
  Works naturally with:
  – **FD** (Forked Drivethrough) for queue sidings with overtaking lanes
  – **SR/SS** for route handoff

Vehicle Parking (VP) Module – Structure & Components

1. Purpose of the VP-PKW Module

The VP (Vehicle Parking) module manages a group of parking slots designed for PKW (passenger cars).
It routes vehicles from a shared entry area into one of several parking slots, monitors their parked duration, and manages the controlled exit of vehicles using reversing tracks and final exit hops.
The module provides:

- Controlled single-vehicle exit (exitPending lock)
- Timed parking (per-vehicle MaxParkingDuration)
- Bypass and drive-through handling
- Automatic safety recovery for stuck vehicles
- Manual mode with selective release

---

2. Core Concepts

Each parking slot has a complete routing definition that includes:

- **Entry Target**
  Contact(s) where a vehicle begins the parking process.
- **Bypass Route**
  Contact used when no slot is available or when waiting conditions block entry.
- **Drive-Through Route**
  A straight-through path for vehicles that should not enter the parking area.
- **Route To Slot**
  A list of contacts guiding a vehicle into its assigned slot.
- **Slot Track**
  The physical track where the vehicle stops and remains parked.
- **Entry Wait Contact**
  Logical queue/stop-line contact to enforce single-file entry.
- **Route To Reverse**
  Contacts used to start reversing out of a slot.
- **Reverse Track**
  Short backward track where reversing physically takes place.
- **Final Exit Target**
  Forward-driving exit route after reversing is finished.
- **Alternate Exit Target**
  Optional alternative forward exit path.

All slot definitions are generated from a compact **slotTemplate** list using wildcard expansion.

---

3. Module Variables

The VP module stores all runtime and configuration data under modul.variables.

Primary Module Variables

Variable	Description
VP-Slots	Expanded slot definition table generated from slotTemplates.
VP-SlotStatus	Runtime status table: free, reserved, occupied, or reversing for each slot.
VP-ExitPending	Indicates whether a vehicle is currently exiting (only one allowed at a time).
IsInitialised	True after all templates, validation, and options are set.

Variable	Description
OptionSelectors	List of switch object names (Start, Stop, Clear, Fill, Timing).
OptionDisplay	List of text-field objects used to display current option states.
ModulKeyword	Keyword applied to all objects belonging to this VP module.
ModulName	Name of the module instance (e.g., "VP_1").

4. Per-Vehicle Variables

Each vehicle interacting with the VP system may contain:

Variable	Description
VP_SlotIndex	Index of the assigned parking slot (1–8).
VP_Parked	Whether the vehicle is currently in a slot.
VP_ParkedMs	Accumulated parked time in milliseconds.
VP_FirstExitHop	First hop the vehicle must reach when exiting.
VP_EntryWaitContact	Assigned WAIT contact for this vehicle.
VP_TargetSpeed	Speed restored after releasing from WAIT.
VP_ReverseMode	True while the vehicle is reversing out.
VP_ExitRoute	Forward route assigned after reversing is completed.

5. Initialisation Flow

Initialisation happens once per module instance and performs the following steps:

1. **Assign ModulName and ModulKeyword.**
   These are used for tagging all related contacts, tracks, and displays.
2. **Define Slot Templates.**
   Two template ranges:
   - Slots **1–4**: direct entry
   - Slots **5–8**: indirect entry via PB_1_Entry_2
3. **Expand Templates → VP-Slots.**
   VP_ExpandSlotTemplates converts wildcard definitions into concrete per-slot tables.
4. **Validate All Objects.**
   VP_ValidateSlots ensures every referenced contact or track exists.
5. **Build Runtime Slot Status Table.**
   VP_InitSlotStatus creates one status entry for each slot (free, reserved, etc.).
6. **Assign Keywords to All Involved Objects.**
   Contacts, tracks, and access routes are tagged via VP_SetKeywordsForSlots and VP_AssignKeywordToList.
7. **Validate Option Switches and Displays.**
   Ensures Start / Stop / Fill / Clear / Timing switches and display fields exist.
8. **Initialise Options and Update Display.**
   InitVpOptions sets default modes; UpdateVpOptionDisplay writes them to the layout text fields.
9. **Start the VP_Tick Timer.**
   Repeats every 10 ms (as defined in init), enabling full VP system operation.

6. Slot Template Model

Each slot generated from templates contains:

Routing & Behaviour

- entryTarget
- bypass
- driveThrough
- routeToSlot
- routeToReverse
- finalExitTarget
- alternateExitTarget

Track Resources

- slotTrack
- reverseTrack
- roadAccess

Logic Contacts

- entryWaitContact
- exitStop

This creates a complete behaviour package for each slot from entry → parking → reverse → forward exit.

---

7. Switch System

The VP module supports manual interactions:

Start

- Starts the VP runtime (VP_Start)
- Begins the VP_Tick loop (unless timingMode = off)

Stop

- Stops the VP_Tick loop and all processing

Release (Manual Mode)

- Only active when timingMode = "manual"
- Selects the next vehicle using VP_SelectVehicleForManualRelease
- Performs VP_ReleaseVehicle for exactly one slot

TFill, TClear, TTiming

- Cycle option modes for fillMode, clearMode, timingMode
- Display updates automatically via UpdateVpOptionDisplay

---

8. Behavioural Overview (Summary)

- Vehicles begin at **entryTarget** → evaluated for eligibility.
- If **exitPending** is true, entry traffic is held at **entryWaitContact**.
- Valid vehicles are routed through **routeToSlot** into their slot.
- Park timers increment every VP_Tick.
- When a slot expires, **VP_ReleaseVehicle** triggers reverse+exit flow.
- After the first exit hop, **VP-ExitPending** clears and entry restarts.
- Safety functions ensure stuck vehicles or missing contacts do not break the system.

VP Component Definition

```
local slotTemplates = {
        {
        index = "1-4",
        entryTarget = {"PB_1_Entry" },
        bypass = {"PB_1_Bypass" },
        driveThrough = {"PB_1_Exit_Alternative" },
        allowedTypes = {"PKW"},
        -- Contacts used to drive INTO the slot (sequence of contacts)
        routeToSlot = {"PB_1_TSlot_*" },      -- slot target
        -- Track that represents the actual parking location
        slotTrack = "PB_1_SSlot_*",
        entryWaitContact = "PB_1_Entry_Wait",
        -- Contacts used to create target for reversing OUT of the slot
        routeToReverse = { "PB_1_Revers_1"},
        reverseTrack = "PB_1_RSlot_*",
          -- Forward Exit
        finalExitTarget = {"PB_1_ExHop0","PB_1_Exit"},
        alternateExitTarget = {"PB_1_ExHop0","PB_1_Exit_Alternative"},
        roadAccess = {"PB_1_Road_Access1", "PB_1_Road_Access2"},
        exitStop={"PB_1_Exit_Stop"}
        },
        {
        index = "5-8",
        entryTarget = {"PB_1_Entry" },
        bypass = {"PB_1_Bypass" },
        driveThrough = {"PB_1_Exit_Alternative" },
        allowedTypes = {"PKW"},
        -- Contacts used to drive INTO the slot (sequence of contacts)
        routeToSlot = {"PB_1_Entry_2" ,"PB_1_TSlot_*" },      -- slot target
        -- Track that represents the actual parking location
        slotTrack = "PB_1_SSlot_*",
        entryWaitContact = "PB_1_Entry_Wait",
        -- Contacts used to create target for reversing OUT of the slot
        routeToReverse = { "PB_1_Revers_2"},
        reverseTrack = "PB_1_RSlot_*",
        -- Forward Exit
        finalExitTarget = {"PB_1_ExHop0","PB_1_Exit"},
        alternateExitTarget = {"PB_1_ExHop0","PB_1_Exit_Alternative"},
        roadAccess = {"PB_1_Road_Access1", "PB_1_Road_Access2"},
        exitStop={"PB_1_Exit_Stop"}
        }
    }


      -- Switch Names
      modul.variables["OptionSelectors"]  =
        { "VP_1-TFill", "VP_1-TClear", "VP_1-TTiming", "VP_1-Start", "VP_1-Stop", "VP_1-Release"}
      -- Display Names
      modul.variables["OptionDisplay"]    = {"VP_1-DFill", "VP_1-DClear", "VP_1-DTiming"}
```

🚗 VP Slot Lifecycle Diagram

```
                         (initial state)
                  +-------------------------+
                  |        FREE SLOT        |
                  |   slotStatus = "free"   |
                  |   vehicle = nil         |
                  +-------------------------+
                              |
                              ▼
  ENTRY CONTACT (vehicle hits entryTarget)
  VP_HandleEntryFlow()
  VP_HandleEntryContact()
 ______________________________________________
 • Choose slot (VP_FindSlotForVehicle)
 • Reserve slot
 • Build routeToSlot (drive hops)
 • Set VP_SlotIndex on vehicle
 ______________________________________________

                              |
                              ▼
                  +-------------------------+
                  |      RESERVED SLOT      |
                  | slotStatus = "reserved" |
                  |  vehicle = <veh.name>   |
                  +-------------------------+
                              |  physical routeToSlot hops
                              ▼
                     SLOT ARRIVAL CONTACT
                       VP_HandleSlotArrival()
 ______________________________________________________________
 • Slot becomes OCCUPIED
 • VP_ParkedMs reset to 0
 • vehicle.target cleared
 • slotStatus.status = "occupied"
 ______________________________________________________________
                              | parked
                              ▼
                  +-------------------------+
                  |      OCCUPIED SLOT      |
                  |  slotStatus = occupied  |
                  |  VP_ParkedMs = timer    |
                  +-------------------------+
                              |  VP_Tick increments VP_ParkedMs
                              ▼
                    VP_RunSlotProcessingLoop()
VEHICLE PARKING TIME EXPIRED (VP_ParkedMs >= MaxParkingDuration)
 ______________________________________________________________
 • VP_ReleaseVehicle()
 • Clear VP state on vehicle
 • Build exit route (routeToReverse + exit hops)
 • Set VP_FirstExitHop on vehicle
 • Set VP-ExitPending = true
 ______________________________________________________________
                              | vehicle drives reverseTrack chain
                              ▼
                 REVERSE END (last reverseTrack hop)
                    kind = "reverseEnd"
 ______________________________________________________________
 • VP-ExitPending = true (kept active)
 • Next: vehicle drives exit hops (ExHop0..n)
 ______________________________________________________________
                              |
                              |---------------► EXIT BLOCK POSSIBLE
                              |                  (ExitStop)
                              ▼
                FIRST EXIT HOP CONTACT (VP_FirstExitHop)
                    VP_HandleExitHop()
 ______________________________________________________________
 • VP-ExitPending = false
 • Clear VP_ParkedMs / VP_Parked
 • Free slot immediately
 • Release waiting vehicle or entry blocked vehicle
 ______________________________________________________________
                              | downstream exit hops
                              ▼
                     EXIT CONTACT (final)
                       VP_HandleExit()
 ______________________________________________________________
 • Defensive cleanup
 • Clears VP state if anything remains
 ______________________________________________________________
                              ▼
                  +-------------------------+
                  |        FREE SLOT        |
                  |  slotStatus = "free"    |
                  |  vehicle = nil          |
                  +-------------------------+
```

🧩 State Summary

FREE

- Idle slot, available for parking.
- Entered via:
  - Startup initialisation
  - End of exit flow (VP_HandleExitHop or VP_HandleExit)
  - Housekeeping fix

RESERVED

- Slot chosen but vehicle not yet arrived.
- Created by: VP_HandleEntryContact → VP_SendVehicleToSlot.

OCCUPIED

- Vehicle physically at slotTrack.
- Created by: VP_HandleSlotArrival.

RELEASING

(Not stored as explicit state — but conceptually a phase)

Triggered by:

- VP_RunSlotProcessingLoop → VP_ReleaseVehicle.

Internals:

- ReverseTrack hops
- VP_FirstExitHop set
- VP-ExitPending = true

EXITING

- Vehicle driving exit hops.
- Ends at VP_HandleExitHop then VP_HandleExit.

🔍 Diagnostic: Where each state is touched

Transition	**Function(s)**
FREE → RESERVED	VP_HandleEntryContact
RESERVED → OCCUPIED	VP_HandleSlotArrival
OCCUPIED → RELEASING	VP_RunSlotProcessingLoop + VP_ReleaseVehicle
RELEASING → EXITING	reverseTrack + exit hops
EXITING → FREE	VP_HandleExitHop / VP_HandleExit
Any → FREE	VP_Housekeeping (repair)

```
VPContactTrigger
│
├── VP_IsDuplicateContactHit
├── VP_HandleExitHop
│   ├── free slot via VP_SetSlotStatusFromVehicle
│   ├── clear VP-ExitPending
│   ├── clear VP_FirstExitHop
│   ├── release waiting vehicle
│   └── release entry-blocked vehicle
├── VP_ClassifyContact
│   ├── detect WAIT via vehicle.VP_EntryWaitContact
│   ├── scan def.entryTarget
│   ├── scan def.routeToSlot[last]
│   ├── scan def.finalExitTarget[last]
│   ├── scan def.alternateExitTarget[last]
│   ├── scan def.bypass
│   ├── scan def.driveThrough
│   ├── scan def.exitStop
│   └── detect reverseEnd via def.routeToReverse[last]
└── dispatch by kind:
    │
    ├── "wait"
    │      └── VP_HandleWaitContact
    │            ├── read VP-ExitPending
    │            ├── read VP-EntryPending
    │            ├── stop vehicle + store VP_WaitingVeh (if blocked)
    │            └── clear VP_WaitingVeh (if free)
    ├── "entry"
    │      └── VP_HandleEntryFlow
    │            ├── if VP_WaitingVeh exists → VP_AssignBypass
    │            ├── if no MaxParkingDuration → VP_AssignBypass
    │            ├── VP_HandleEntryPrecheck
    │            │      ├── check vehicle.VP_SlotIndex
    │            │      ├── repair stale assignments
    │            │      └── scan slotStatus by vehicle name
    │            └── VP_HandleEntryContact
    │                   ├── VP_IsWaitBlocked
    │                   ├── VP_ClearVehicleState
    │                   ├── VP_FindSlotForVehicle
    │                   │      ├── VP_GetSlotsReachableFromEntryContact
    │                   │      ├── VP_SlotAllowsVehicleType
    │                   │      └── fillMode selection
    │                   ├── VP_SendVehicleToSlot
    │                   │      ├── build hop list from def.routeToSlot
    │                   │      └── assign vehicle.target
    │                   ├── set VP_SlotIndex
    │                   └── set VP_EntryWaitContact
    ├── "slot"
    │      └── VP_HandleSlotArrival
    │            ├── mark st.status="occupied"
    │            ├── store st.vehicle = vehicle.name
    │            ├── vehicle.VP_ParkedMs = 0
    │            ├── vehicle.VP_SlotIndex = idx
    │            ├── vehicle.target = nil
    │            └── vehicle.currentSpeed = 0
    ├── "reverseEnd"
    │      └── VP_HandleReverseEnd
    │            ├── vehicle.drivingDirection = 1
    │            ├── unwrap vehicle.VP_ExitRoute
    │            ├── build exit hop entity list
    │            ├── vehicle.target = hops
    │            └── clear reverse/exit vars
    ├── "exit"
    │      └── VP_HandleExit
    │            ├── read vehicle.VP_SlotIndex
    │            ├── read slotStatus[idx]
    │            ├── CASE A: slot already free → cleanup vehicle
    │            ├── CASE B: still occupied by same vehicle → free slot
    │            └── CASE C: inconsistent → warn + cleanup vehicle
    ├── "exitStop"
    │      └── VP_AssignBypass or tick-safety clearing
    │
    ├── "bypass"
    │      └── VP_AssignBypass
    │            ├── get bypassList[1]
    │            └── vehicle.target = bypassTarget
    └── "drive"
           └── VP_AssignBypass (same flow)
```

VP_Tick

```
│
├── VP_EntryQueueTick
│     ├── for each slot:
│     ├── if st.status == "reserved":
│     │      ├── read def.slotTrack
│     │      ├── layout:getVehiclesOn(track)
│     │      ├── compare veh.name == st.vehicle
│     │      └── promote st.status → "occupied"
│     └── end
│
├── VP_ExitMonitorTick (not in provided file, skip)
│
├── VP_RunSlotProcessingLoop
│     ├── VP_LogSlotOverview
│     │
│     ├── for each slot:
│     │     ├── if st.status == "occupied":
│     │     │     ├── track = layout:getEntityByName(def.slotTrack)
│     │     │     ├── vehList = layout:getVehiclesOn(track)
│     │     │     ├── if no physical vehicle:
│     │     │     │     └── free slot + continue
│     │     │     ├── if mismatch st.vehicle != veh.name:
│     │     │     │     └── repair st.vehicle
│     │     │     └── VP_ProcessOccupiedSlot
│     │     │           ├── parked = veh.VP_ParkedMs + tickMs
│     │     │           ├── if parked >= MaxParkingDuration:
│     │     │           │      └── VP_ReleaseVehicle
│     │     │           │             ├── lookup vehicle by st.vehicle
│     │     │           │             ├── set reverse mode vars
│     │     │           │             ├── build hops from def.routeToReverse
│     │     │           │             ├── vehicle.target = hops
│     │     │           │             ├── vehicle.currentSpeed = -5
│     │     │           │             └── st.status = "reversing"
│     │     │           └── return (end tick)
│     │     └── else skip
│     │
│     └── end loop
│
└── (safety logic done outside VP_Tick in your setup):
      │
      ├── VP_Housekeeping
      │     ├── verify physical occupancy
      │     ├── auto-free stale slots
      │     ├── restore vehicle slot mappings
      │     └── repair parked flags
      │
      ├── VP_Safety_CheckWaitingVeh
      │     ├── physical presence test
      │     ├── if empty → clear VP_WaitingVeh
      │     ├── if vehicle present & no exitPending → start it
      │     └── else wait
      │
      ├── VP_Safety_RecalculateAccessBlockedCount
      │     ├── scan roadAccess tracks
      │     ├── count physical blockers
      │     └── write VP_AccessBlockedCount
      │
      ├── VP_Safety_ClearExitStopWaitIfEmpty
      │     ├── detect exitStop[1]
      │     ├── if empty → clear VP_ExitStopWaitingVeh
      │     └── done
      │
      ├── VP_Safety_ExitTimeoutAndEntryRecovery
      │     ├── VP_HandleExitTimeout
      │     │     ├── increment VP-ExitTicks
      │     │     ├── if >= timeout → reset exit flow
      │     │     └── restart stuck waiting vehicle
      │     └── VP_StartEntryIfWaitEmpty
      │           ├── find stopped vehicle on entryContact
      │           ├── call VP_HandleEntryContact
      │           └── restart it
      │
      └── VP_Safety_FreeReversingSlots
            ├── for every reversing slot:
            ├── if reverseTrack physically empty:
            └── free slot
```

Init Phase (runs once if modul.variables["IsInitialised"] ~= true)

```
└─ (top-level init block)
   ├─ set modul.variables["ModulKeyword"]
   ├─ set modul.variables["ModulName"]
   ├─ define slotTemplates
   │
   ├─ VP_ExpandSlotTemplates
   │   ├─ VP_ParseIndexSpec
   │   ├─ VP_ReplaceWildcard
   │   ├─ flatten MBS object lists
   │   └─ wildcard expansion for all list fields
   │
   ├─ modul.variables["VP-Slots"] = result
   │
   ├─ VP_ValidateSlots
   │   ├─ check slotTrack exists
   │   ├─ check reverseTrack exists
   │   ├─ checkList(driveThrough)
   │   ├─ checkList(alternateExitTarget)
   │   ├─ checkList(bypass)
   │   ├─ checkList(entryTarget)
   │   ├─ checkList(routeToSlot)
   │   ├─ checkList(routeToReverse)
   │   └─ checkList(finalExitTarget)
   │
   ├─ VP_InitSlotStatus
   │   └─ allocate status[i] = {status="free", vehicle=nil, ...}
   │
   ├─ modul.variables["VP-SlotStatus"] = vpStatus
   │
   ├─ VP_SetKeywordsForSlots
   │   └─ apply() per:
   │          slotTrack
   │          reverseTrack
   │          driveThrough[*]
   │          alternateExitTarget[*]
   │          bypass[*]
   │          entryTarget[*]
   │          routeToSlot[*]
   │          routeToReverse[*]
   │          finalExitTarget[*]
   │          roadAccess[*]
   │
   ├─ VP_AssignKeywordToList (slot tracks)
   │   └─ applyKeyword for each name
   │       ├─ remove modulName keyword
   │       ├─ set modulKeyword = keyword
   │       └─ set typeKeyword = keyword
   │
   ├─ VP_AssignKeywordToList (reverse tracks)
   │   └─ same applyKeyword sequence
   │
   ├─ VP_AssignKeywordToList (road access tracks)
   │   └─ same applyKeyword sequence
   │
   ├─ ValidateObjectList (OptionSelectors, modulKeyword)
   ├─ ValidateObjectList (OptionDisplay, modulKeyword)
   ├─ ValidateObjectList (OptionSelectors, modulName)
   ├─ ValidateObjectList (OptionDisplay, modulName)
   │
   ├─ InitVpOptions
   │   └─ set modul.variables["VpOptions"] = {fillMode, clearMode, timingMode}
   │
   ├─ UpdateVpOptionDisplay
   │   ├─ UpdateVpOptionDisplayValue (fill)
   │   ├─ UpdateVpOptionDisplayValue (clear)
   │   └─ UpdateVpOptionDisplayValue (timing)
   │
   ├─ modul.variables["IsInitialised"] = true
   ├─ modul.variables["VP-ExitPending"] = false
   │
   └─ modul.timers["VP_Tick"]:start
```

SwitchEventHandler (VpSwitchEvents)

```
└─ ProcessNextVpOption (if controller.name matches VP_1-TFill, TClear, TTiming)
   ├─ GetVpModulName
   ├─ SetNextVpOptionValue
   │   ├─ NextFillMode
   │   │     └─ VpCycleOptionValue
   │   ├─ NextClearMode
   │   │     └─ VpCycleOptionValue
   │   └─ NextTimingMode
   │         └─ VpCycleOptionValue
   └─ UpdateVpOptionDisplay
       ├─ UpdateVpOptionDisplayValue (fill)
       ├─ UpdateVpOptionDisplayValue (clear)
       └─ UpdateVpOptionDisplayValue (timing)

SwitchEventHandler
└─ (after ProcessNextVpOption)
   ├─ VpIsStart(controller.name)
   ├─ VpIsStop(controller.name)
   ├─ VpIsRelease(controller.name)
   ├─ GetVpModulName
   ├─ vpModul = layout:getEventsByName(vpModulName)[1]
   │
   ├─ if Start:
   │     └─ VP_Start
   │          ├─ VP_InitSlotStatus (only if missing)
   │          └─ timer:start
   │
   ├─ if Stop:
   │     └─ VP_Stop
   │          └─ timer:stop
   │
   └─ if Release:
         ├─ VP_SelectVehicleForManualRelease
         │     ├─ scan VP-SlotStatus for occupied slots
         │     ├─ apply clearMode algorithm
         │     └─ return slotIndex, vehicle
         └─ VP_ReleaseVehicle
               ├─ lookup vehicle via name
               ├─ set reverse mode vars
               ├─ build hops from def.routeToReverse
               ├─ vehicle.target = hops
               ├─ vehicle.currentSpeed = -5
               └─ st.status = "reversing"
```