

**VECTO Software Developer Guide**

**Content**

[1. Introduction 3](#_Toc391476082)

[2. Main classes 3](#_Toc391476083)

[2.1. cMOD 3](#_Toc391476084)

[2.2. cPower 3](#_Toc391476085)

[2.3. cVh 4](#_Toc391476086)

[2.4. cVECTO 4](#_Toc391476087)

[2.5. cVEH 5](#_Toc391476088)

[2.6. cENG 5](#_Toc391476089)

[2.7. cGBX 5](#_Toc391476090)

[2.8. cMAP 5](#_Toc391476091)

[2.9. cDRI 6](#_Toc391476092)

[2.10. cFLD 6](#_Toc391476093)

[2.11. cVSUM 6](#_Toc391476094)

[2.12. cDeclaration 6](#_Toc391476095)

[2.13. cConfig 6](#_Toc391476096)

[3. Global objects 7](#_Toc391476097)

[4. Program flow structure 7](#_Toc391476098)

[4.1. Initialisation 10](#_Toc391476099)

[4.2. Job Loop 10](#_Toc391476100)

[4.2.1. Declaration Mode Initialisation 10](#_Toc391476101)

[4.3. Cycle Loop 11](#_Toc391476102)

[4.4. Loading Loop 11](#_Toc391476103)

[4.4.1. Driving Cycle Pre-processing (M1) 12](#_Toc391476104)

[4.4.2. Driver Pre-processing & Power Calculation 13](#_Toc391476105)

[5. Dependencies 15](#_Toc391476106)

[5.1.1. Software libraries 15](#_Toc391476107)

[5.1.2. Other files 15](#_Toc391476108)

# Introduction

The following guide is targeted to developers who are working on the VECTO source code. It is designed as a starting point to help understanding the structure of the source code and thus only covers the most important classes and functions. The guide is intended as addendum to the LOT3 final report (notably chapter 6.1) whose content is assumed to be known to the developer. It does not include descriptions of the physical models behind the functions or other background information.

Dark red labels (like **M1** in Figure 1) refer to the corresponding modules and functions of the final report's chapter 6.1.

Most methods in VECTO are designed as functions with Boolean return type and return:

* True - if the function was finished without errors
* False - if the function was aborted due to errors

Functions not following this rule are described separately.

# Main classes

The following chapter describes the most important classes in VECTO.

## cMOD

This class is mainly used to collect modal results like engine power and engine speed. Furthermore it contains:

* The methods used by Distance Correction:
  + ***Cut()*** -remove the next time step to reduce travelled distance
  + ***Duplicate()*** -repeat the last time step to increase the travelled distance
* ***FCcalc()*** -calculates fuel consumption based on the FC map and applies the auxiliary power and WHTC correction.
* ***Output()*** -Write modal results file (.vmod)
* Initialisation methods

## cPower

This class contains the core calculation routines. The most important methods are:

* ***Calc()*** -Vehicle longitudinal dynamics simulation. Calculates engine torque and speed for all time steps.
* ***EngCalc()*** -Engine Only simulation. Used to calculate engine power based on the clutch power form the load cylce (.vdri)
* ***PreRun()*** -changes the driving cycle according to the Eco Roll, Overspeed and Look-Ahead Coasting functions.
* ***fCoastingSpeed()*** [overloaded] -returns engine motoring vehicle speed for the given time step
* ***fRolloutSpeed()*** -returns roll out vehicle speed for the given time step.
* Gear Shift Methods
  + ***fGearVECTO()*** -Main gear shift method. Returns gear number
  + ***fStartGear()*** -returns gear for first time step or after vehicle stand still
  + ***fGearTC()*** -returns gear for vehicles with Automatic Transmission and Torque Converter
  + ***fFastGearCalc()*** -Simplified gear shift method used for PreRun. Returns gear number
* Functions to calculate power form wheel to engine most notably:
  + ***fPwheel()*** [overloaded] -returns power demand at wheels for the given time step.
  + ***fPaux()*** -returns total auxiliary power demand at the given time step, including the optionally defined power demand from the driving cycle

## cVh

This class holds driving cycle (mission profile) information like:

* speed profiles (target and actual speed)
* altitude for road gradient calculation
* Actual driven distance for Distance Correction
* Additional (optional) cycle input like gears

Furthermore it holds methods for:

* Speed reduction (full load iteration)
* Acceleration limiting

## cVECTO

This class represents the Job File (.vecto) and includes methods for file in- and output. Other important methods:

* ***AuxInit()*** -initialises auxiliaries and triangulates loss maps
* ***PauxSum()*** -returns total auxiliary power demand for the given time step
* ***aDesMax()*** -returns the acceleration limit (.vacc) at a given vehicle speed
* ***aDesMin()*** -returns the deceleration limit (.vacc) at a given vehicle speed
* ***DeclInit()*** -sets generic parameters for Declaration mode

## cVEH

This class represents the Vehicle File (.vveh) and includes methods for file in- and output. Other important methods:

* ***Cd()*** -returns drag coefficient including cross wind correction
* ***RtPeLoss()*** -returns retarder loss torque
* ***DeclInit()*** -sets non-loading-dependent generic parameters for Declaration mode
* ***DeclInitLoad()*** -sets loading-dependent generic parameters for Declaration mode

## cENG

This class represents the Engine File (.veng) and includes methods for file in- and output. Other important methods:

* ***Init()*** -reads full load curve(s) (.vfld) and the FC map (.vmap)
* ***DeclInit()*** -sets generic parameters for Declaration mode

## cGBX

This class represents the Gearbox File (.vgbx) and includes methods for file in- and output. Other important methods:

* ***GSinit()*** -initialises the gearbox and reads shift polygons (.vgbs)
* ***TrLossMapInit()*** -reads and triangulates transmission loss maps (.vtlm)
* ***IntpolPeLoss()*** -returns transmission losses for the given gear
* ***TCiteration()*** -calculates the Torque Converter operation point
* ***DeclInit()*** -sets generic parameters for Declaration mode

## cMAP

This class represents the FC map (.vmap) and contains the file input method. Furthermore it includes:

* ***fFCdelaunay\_Intp()*** -returns the interpolated fuel consumption.

## cDRI

This class represents the driving cycle (.vdri) and contains the file input method. Other important methods:

* ***GradToAlt()*** -converts the cycle's road gradient to altitude.
* ***ConvStoT()*** -converts the distance-based cycle to time-based.
* ***FirstZero()*** -duplicates the first time step if its vehicle speed is zero (vehicle stand-still) in order not to lose the zero-value after the new time steps are created.
* ***ConvTo1Hz()*** -converts the time-based cycle to a 1Hz cycle and generates the new time steps. See LOT3 final report for details.

## cFLD

This class represents the full load and drag curve file (.vfld) and contains the file input method. Other important methods:

* ***Pfull()*** [overloaded] -returns the full load power at the given engine speed, optionally using the PT1 transient correction
* ***Pdrag()*** -returns the full load power at the given engine speed
* ***DeclInit()*** -sets generic parameters for Declaration mode

## cVSUM

This class represents the summary/average results file (.vsum and .vsum.json) and contains the file output methods for both files.

## cDeclaration

This class contains all methods used for Declaration mode in order to load the vehicle segment table and other generic parameters. It includes several functions which return generic parameters to each input file class. Furthermore it includes the methods to create the pdf report in the nested ***cReport*** class.

## cConfig

This class represents the application configuration file and includes methods for file in- and output.

## GUI\_Subs

This module contains functions for interaction between the *BackgroundWorker* and the GUI. The most important function is:

* ***WorkerMsg()*** -sends a message, warning or error to the main form.

Note that the *BackgroundWorker* cannot interact with GUI controls directly!

# Global objects

Most classes described in the previous chapter are used via a single global instance as listed below:

* ***VEC*** -Global instance of ***cVECTO***. Currently used job file.
* ***VEH*** -Global instance of ***cVEH***. Currently used vehicle
* ***ENG*** -Global instance of ***cENG***. Currently used engine
* ***GBX*** -Global instance of ***cGBX***. Currently used gearbox
* ***MAP*** -Global instance of ***cMAP***. Currently used FC map
* ***DRI*** -Global instance of ***cDRI***. Currently used driving cycle
* ***FLD*** -Global list which includes one instance of ***cFLD*** for each gear in ***GBX***.
* ***MODdata*** -Global instance of ***cMOD***.
* ***VSUM*** -Global instance of ***cVSUM***.
* ***Cfg*** -Global instance of ***cConfig***.
* ***Declaration*** -Global instance of ***cDeclaration***.

The instances are declared during application start up and the vehicle-related instances are initialised in ***M\_MAIN.ReadFiles()***.

# Program flow structure

This chapter describes the main program flow for running a calculation for a set of job files (.vecto).

***M\_MAIN.VECTO()***

-is the main routine which is launched via a *BackgroundWorker* from the main GUI form. It returns a value of the type *tCalcResult*:

* Err - The calculation was aborted due to an error.
* Abort - The calculation was aborted by the user.
* Done - The calculation finished successfully. This only means the main routine was not aborted. Errors might still occur for individual job files.

The main routine consists of three nested loops:

* The **Job Loop** runs through all job files uses for the current calculation
* The **Cycle Lop** runs through all cycles used for each job file. In Batch Mode the same cycle list is used for all jobs otherwise each job has its own job list.
* The **Loading Loop** is used to loop through all vehicle loadings (full, empty and reference loading) in Declaration Mode.

Figure 1 shows the structure. Note that **M1** to **M4** refer to the main VECTO modules as described in the final report.



Figure 1: Main VECTO routine

## Initialisation

***VSUM.Init()***

-initialises the sum/average results file (VSUM).

## Job Loop

The **Job Loop** runs through all job files uses for the current calculation.

***M\_MAIN.ReadFiles()***

-reads the job file (.vecto) and all secondary input files (e.g. .vgbx). The driving cycle(s) are read later in the Cycle Loop.

### Declaration Mode Initialisation

This block is only executed in Declaration Mode. It is used to initialise the pdf report file and to calculate the WHTC results for the WHTC Correction which is applied after the fuel consumption calculation.



Figure 2 Declaration Mode Initialisation

***Declaration.ReportInit()***

-initialises the pdf report file, adding vehicle data, etc.

***MODdata.Init()***

-creates/clears all the list objects used to collect modal results.

***Declaration.WHTCinit()***

-loads the WHTC cycle for calculation in Engine Only Mode.

***MODdata.CycleInit()***

-is part of the initialisation process.

***MODdata.Px.Eng\_Calc()***

-runs the Engine Only Calculation to provide modal results for engine torque and speed for FC interpolation.

***MODdata.FCcalc()***

-interpolated FC based on engine speed and torque.

***Declaration.WHTCcorrCalc()***

-calculates the WHTC results for the Urban, Rural and Motorway parts.

## Cycle Loop

The **Cycle Lop** runs through all cycles used for each job file. In Batch Mode the same cycle list is used for all jobs otherwise each job has its own job list.

***Declaration.CalcInitCycle()***

-sets generic input data for Declaration Mode which depends on the Mission Profile

## Loading Loop

The **Loading Loop** is used to loop through all vehicle loadings (full, empty and reference loading) in Declaration Mode.

***Declaration.CalcInitLoad()***

-sets generic input data for Declaration Mode which depends on the Mission Profile and Loading

***MODdata.Init()***

-creates/clears all the list objects used to collect modal results.

### Driving Cycle Pre-processing (M1)



Figure 3: Driving Cycle Pre-processing (M1)

***DRI.ReadFile()***

-reads the driving cyle (mission profile)

***DRI.GradToAlt()***

-converts the cycle's road gradient to altitude.

***MODdata.Vh.SetAlt()***

-generates the altitude-over-distance data

***DRI.ConvStoT()* M1.1**

-converts the distance-based cycle to time-based.

***DRI.FirstZero()***

-duplicates the first time step if its vehicle speed is zero (vehicle stand-still) in order not to lose the zero-value after the new time steps are created.

***DRI.ConvTo1Hz()* M1.2, M1.3**

-converts the time-based cycle to a 1Hz cycle. Here the new time steps are created. See LOT3 final report for details.

***MODdata.CycleInit()***

-is part of the initialisation process, e.g. prepares gear or engine speed input from driving cycle, etc.

### Driver Pre-processing & Power Calculation



Figure 4: Driver Pre-processing & Power Calculation

***VEC.AuxInit()***

-reads auxiliary input files and triangulates aux loss maps. Checks if driving cycle provides supply power for each auxiliary (if not using Declaration Mode).

***VEH.VehmodeInit()***

-initialises all vehicle sub-components:

* Reads Retarder loss-torque file
* Reads and triangulates transmission loss maps.
* Calculates RRC
* Reads cross wind correction file

***GBX.TCinit()***

-reads the torque converter input file

***MODdata.Px.PreRun()* M2**

-changes the driving cycle according to the Eco Roll, Overspeed and Look-Ahead Coasting functions.

***MODdata.Vh.DistCorrInit()***

-initialises the speed-over-distance data used for the Distance Correction.

***MODdata.Px.Calc()* M3**

-calculates engine torque and speed for all time steps.

***MODdata.CylceKin.Calc()***

-calculates various speed/acceleration-dependent results like stop- and cruising-times, average accelerations, etc.

***MODdata.FCcalc()* M4**

-calculates fuel consumption based on the FC map and applies the auxiliary power and WHTC correction.

***MODdata.Output()***

-writes the modal results file (.vmod).

***Declaration.ReportAddResults()***

-adds the current results to the pdf report.

***VSUM.WriteVSUM(***

-adds the current results to the sum/average results file.

***MODdata.CleanUp()***

-clears modal result lists for the next calculation

***Declaration.WriteReport()***

-finalises the report and writes the pdf file

***VSUM.WriteJSON()***

-writes the sum/average results file in JSON format.

# Dependencies

### Software libraries

The following libraries must be added to the project before it can be compiled:

* **vectolic.dll** -licensing functions
* **itextsharp.dll** -used to generate pdf files
* **Newtonsoft.Json.dll** -for JSON file in- and output

### Other files

The following folders and files must be included in the application folder in order to run VECTO:

* **User Manual** (folder)
* **Declaration** (folder)
* **User Manual\Release Notes.pdf**
* **vectolic.dll**
* **itextsharp.dll**
* **Newtonsoft.Json.dll**
* **license.dat**

All folders and files, except **license.dat** are tracked in Git together with the source code.