wltp Documentation

Release 0.0.8-alpha.2

Kostis Anagnostopoulos

December 01, 2014

Contents

Version 0.0.8-alpha.2

Home https://github.com/ankostis/wltp

Documentation https://wltp.readthedocs.org/

PyPI https://pypi.python.org/pypi/wltp

TravisCI https://travis-ci.org/ankostis/wltp

License EUPL 1.1+

A calculator of the gear-shifts profile for light-duty-vehicles (cars) according to *UNECE* draft of the *WLTP*.

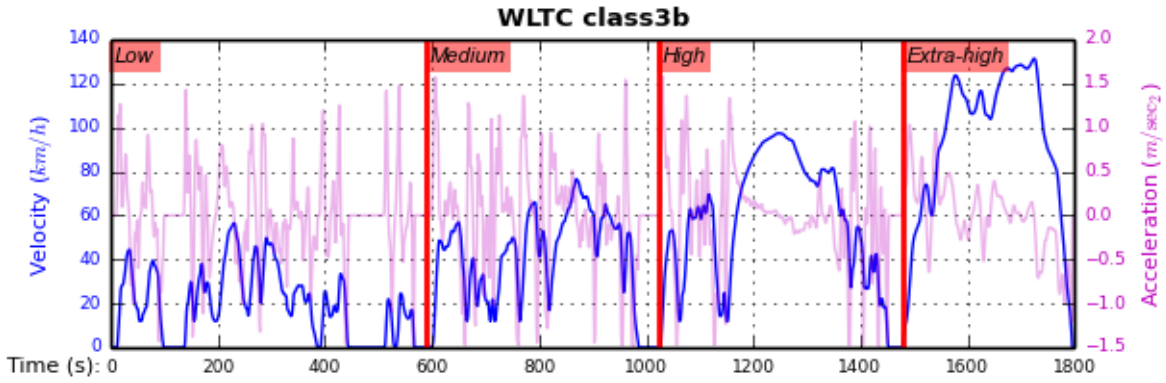


Figure 1: **Figure 1:** *WLTP cycle for class-3b Vehicles*

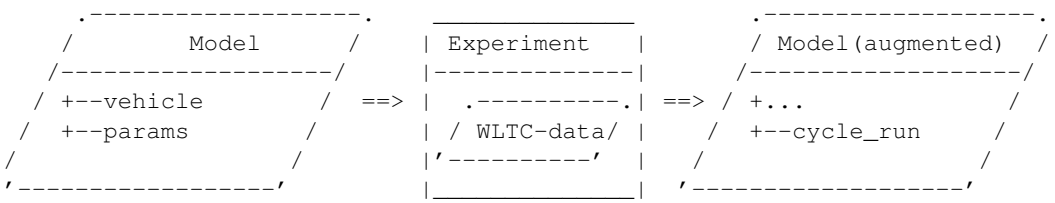
Important: This project is still in *alpha* stage. Its results are not considered "correct", and no approval procedure should rely on them. Some of the known deficiencies are described in *Changes*.

CHAPTER 1

Introduction

The calculator accepts as input the vehicle-specifications and parameters for modifying the execution of the *WLTC* cycle and spits-out the it gear-shifts of the vehicle, the attained speed-profile, and any warnings. It certainly does not calculate any CO2 emissions or other metrics.

An "execution" or a "run" of an experiment is depicted in the following diagram:



1.1 Install

Requires Python 3.3+.

Tip: WinPython and Anaconda python distributions for *Windows* and *OS X*, respectively.

You can install the project directly from the PyPI repository with **pip**. Notice that `--pre` is required, since all realeased packages so far were *pre*-release (`-alpha`) versions:

```
$ pip3 install wltp
```

Or you can build it from the latest sources (assuming you have a working installation of git):

```
$ git clone https://github.com/ankostis/wltp.git wltp
$ cd wltp
$ python3 setup.py install .
```

That way you get the complete source-tree of the project:

```
+--wltp/            ## (package) The python-code of the calculator
|   +--cycles/      ## (package) The python-code for the WLTC data
|   +--test/        ## (package) Test-cases and the wltp_db
|   +--model        ## (module) Describes the data for the calculation
|   +--experiment   ## (module) The calculator
+--docs/            ## Documentation folder
+--util/            ## Scripts for preprocessing WLTC data and the wltp_db
+--wltp.py          ## (script) The cmd-line entry-point script for the calculator
```

```
+--README.rst
+--CHANGES.rst
+--LICENSE.txt
```

1.2 Python usage

Here is a quick-start example:

```
>>> from wltp import model
>>> from wltp.experiment import Experiment
>>> import json                  ## Just for pretty-printing model

>>> mdl = {
    "vehicle": {
        "mass":     1500,
        "v_max":    195,
        "p_rated":  100,
        "n_rated":  5450,
        "n_idle":   950,
        "n_min":    None, # Can be overriden by manufacturer.
        "gear_ratios":      [120.5, 75, 50, 43, 37, 32],
        "resistance_coeffs":[100, 0.5, 0.04],
    }
}

>>> processor = Experiment(mdl)
>>> mdl = processor.run()
>>> print(json.dumps(mdl['params'], indent=2))
{
  "f_n_min_gear2": 0.9,
  "v_stopped_threshold": 1,
  "wltc_class": "class3b",
  "f_n_min": 0.125,
  "f_n_max": 1.2,
  "f_downscale": 0,
  "f_inertial": 1.1,
  "f_n_clutch_gear2": [
    1.15,
    0.03
  ],
  "f_safety_margin": 0.9
}
```

To access the time-based cycle-results it is better to use a `pandas.DataFrame`:

```
>>> import pandas as pd
>>> df = pd.DataFrame(mdl['cycle_run'])
>>> df.columns
Index(['clutch', 'driveability', 'gears', 'gears_orig', 'p_available', 'p_required', 'rpm', 'rpm_norm
>>> df.index.name = 't'
>>> print('Mean engine_speed: ', df.rpm.mean())
Mean engine_speed:  1917.0407829

>>> print(df.head())
  clutch driveability  gears  gears_orig  p_available  p_required  rpm  \
t
```

```
0  False                   0           0            9           0  950
1  False                   0           0            9           0  950
2  False                   0           0            9           0  950
3  False                   0           0            9           0  950
4  False                   0           0            9           0  950

   rpm_norm  v_class   v_real  v_target
t
0         0        0  29.6875         0
1         0        0  29.6875         0
2         0        0  29.6875         0
3         0        0  29.6875         0
4         0        0  29.6875         0

[5 rows x 11 columns]

>>> print(processor.driveability_report())
...
  12: (a: X-->0)
  13: g1: Revolutions too low!
  14: g1: Revolutions too low!
...
  30: (b2(2): 5-->4)
...
  38: (c1: 4-->3)
  39: (c1: 4-->3)
  40: Rule e or g missed downshift(40: 4-->3) in acceleration?
...
  42: Rule e or g missed downshift(42: 3-->2) in acceleration?
...
```

For information on the model-data, check the schema:

```
>>> print(json.dumps(model.model_schema(), indent=2))
{
  "properties": {
    "params": {
      "properties": {
        "f_n_min_gear2": {
          "description": "Gear-2 is invalid when N :< f_n_min_gear2 * n_idle.",
          "type": [
            "number",
            "null"
          ],
          "default": 0.9
        },
        "v_stopped_threshold": {
          "description": "Velocity (Km/h) under which (<=) to idle gear-shift (Annex 2-3.3, p71).",
          "type": [
...
```

For more examples, download the sources and check the test-cases found at `/wltp/test`.

1.3 Cmd-line usage

Note: Not implemented in yet.

To get help:

```
$ python wltp --help          ## to get generic help for cmd-line syntax
$ python wltp -M /vehicle     ## to get help for specific model-paths
```

and then, assuming `vehicle.csv` is a CSV file with the vehicle parameters for which you want to override the `n_idle` only, run the following:

```
$ python wltp -v \
    -I vehicle.csv file_frmt=SERIES model_path=/params header@=None \
    -m /vehicle/n_idle:=850 \
    -O cycle.csv model_path=/cycle_run
```

1.4 IPython usage

Note: Not implemented in yet.

CHAPTER 2

Getting Involved

To provide feedback, use github's Issue-tracker.

2.1 Development

Tip: The commands below are given for a *POSIX* environment (*Linux* & *OS X*). They are simple enough and easy to translate into their *Windows* counterparts, but it would be worthwile to install cygwin to get the same environment on *Windows*.

To get involved with development, first you need to download the latest sources:

```
$ git clone https://github.com/ankostis/wltp.git wltp
$ cd wltp
```

It is preferable that you work from within a virtual-environment. Assuming that you have installed `virtualenv` you can then type the following:

```
$ virtualenv -p <PATH_TO_PYTHON_3> ../wltp.venv
$ .  ../wltp.venv/bin/activate
$ python setup.py develop .
```

Check that the sources are in good shape by running the test-cases and check for errors:

```
$ python setup.py test
```

You can now modify the sources and rerun the tests to ensure that you didn't break anything. If there are no problems, commit them with a usefull message. Split the functionality you want to implement in small well-defined commits, and provide test-cases. If you made a rather important modification, update also the *Changes* file and/or other documents. To see the rendered results of the documents, issue the following command and check the result html file at `build/sphinx/html/index.html`:

```
$ python setup.py build_sphinx
```

When you are finished, push the changes upstream to github and make a *merge_request*. You can check whether your merge-request passed the tests by checking the status of the TravisCI integration-server.

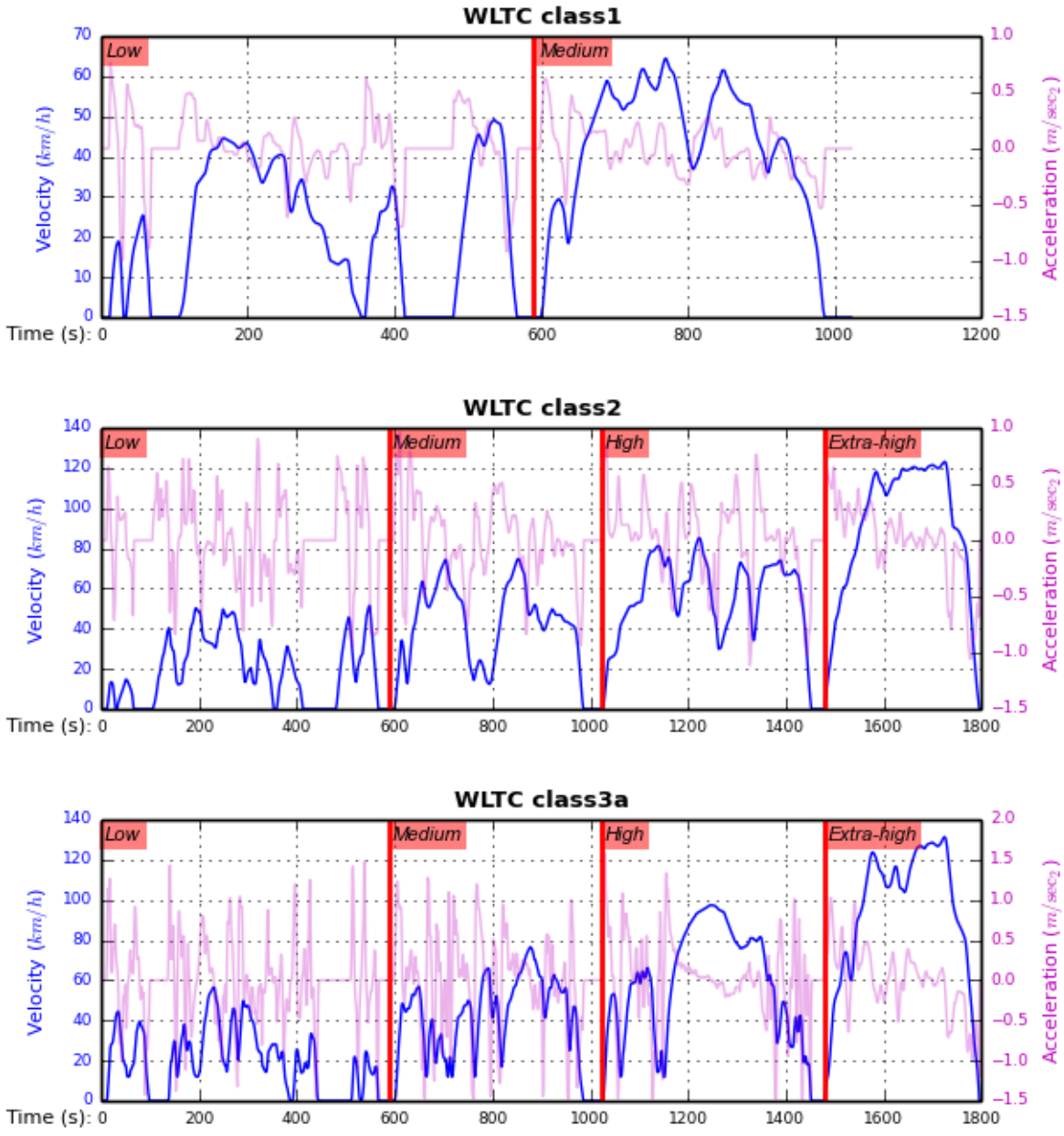
Tip: Skim through the small and excellent IPython developers document: The perfect pull request

2.2 Specs & Algorithm

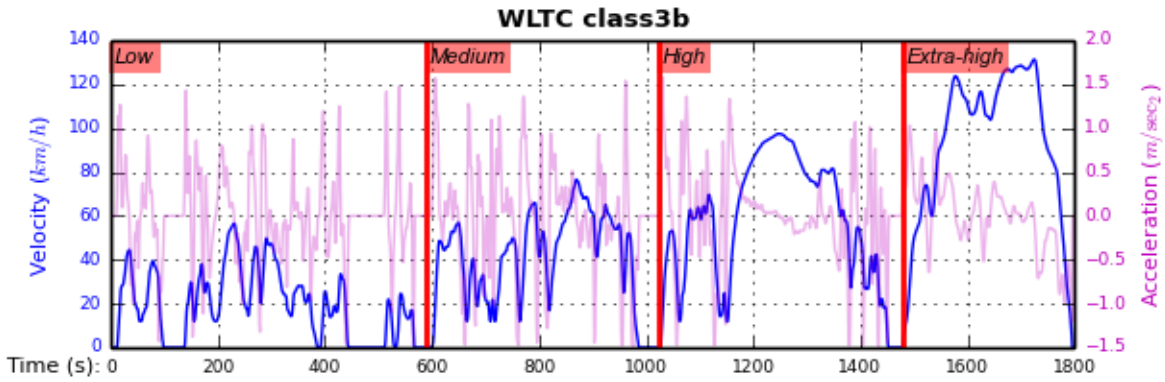
This program was implemented from scratch based on this `GTR specification` (included in the `docs/` dir). The latest version of this *GTR*, along with other related documents can be found at UNECE's site:

- http://www.unece.org/trans/main/wp29/wp29wgs/wp29grpe/grpedoc_2013.html
- https://www2.unece.org/wiki/pages/viewpage.action?pageId=2523179
- Probably a more comprehensible but older spec is this one: https://www2.unece.org/wiki/display/trans/DHC+draft+technical+repo

2.2.1 Cycles



See also:



Changes

2.3 Development team

- **Author:**
  - Kostis Anagnostopoulos
- **Contributing Authors:**
  - Heinz Steven (test-data, validation and review)
  - Georgios Fontaras (simulation, physics & engineering support)
  - Alessandro Marotta (policy support)

CHAPTER 3

Code reference

The core of the simulator is composed from the following modules:

experiment	The core that accepts a vehicle-model and wltc-classes, runs the simulation and updates the model with results (dow
model	The hierarchical data and their schema for the WLTC calculator (defaults, WLTC-data) used by the Model and Expe

Among the various tests, those involved with data and divergence from existing tool are:

experiment_WholeVehicleTests	Tests check top-level functionality.
wltp_db_tests	Compares the results of a batch of wltp_db vehicles against Heinz's tool.
experiment_SampleVehicleTests	Compares the results of synthetic vehicles from JRC against Heinz's tool.

The following scripts in the sources maybe used to preprocess various wltc data:

- util/preprocheinz.py
- util/printwltcclass.py
- util/csvcolumns8to2.py

3.1 Module: wltp.experiment

The core that accepts a vehicle-model and wltc-classes, runs the simulation and updates the model with results (down-scaled velocity & gears-profile).

Attention: The documentation of this core module has several issues and needs work.

ALL_CAPITALS variable denote *vectors*, usually over the velocity-profile (the cycle), for instance, GEARS is like that:

```
 t:||: 0  1  2  3
---+-------------
g1:|[[ 1, 1, 1, 1, ... 1, 1
g2:|   2, 2, 2, 2, ... 2, 2
g3:|   3, 3, 3, 3, ... 3, 3
g4 |   4, 4, 4, 4, ... 4, 4 ]]
```

3.1.1 Vectors

V: floats (#cycle_steps) The wltp-class velocity profile.

GEARS: integers (#gears X #cycle_steps) One row for each gear (starting with 1 to #gears).

GEARS_YES: boolean (#gears X #cycle_steps) One row per gear having `True` wherever gear is possible for each step.

N_GEARS: floats (#gears X #cycle_steps) One row per gear with the Engine-revolutions required to follow the V-profile (unfeasable revs included), produced by multiplying `V * gear-rations`.

See also:

`model` for in/out schemas

since 1 Jan 2014

class `wltp.experiment.`**`Experiment`**(**models*, *skip_model_validation=False*, *validate_wltc=False*)
Runs the vehicle and cycle data describing a WLTC experiment.

See `wltp.experiment` for documentation.

`__init__`(**models*, *skip_model_validation=False*, *validate_wltc=False*)
`model` is a tree (formed by dicts & lists) holding the experiment data.

`skip_model_validation` when true, does not validate the model.

`run`(*overridde_cycle=False*)
Invokes the main-calculations and extracts/update Model values!

@see: Annex 2, p 70

`wltp.experiment.`**`applyDriveabilityRules`**(*V*, *A*, *GEARS*, *CLUTCH*, *ngears*, *driveability_issues*)
@note: Modifies GEARS. @see: Annex 2-4, p 72

`wltp.experiment.`**`calcDownscaleFactor`**(*P_REQ*, *p_max_values*, *downsc_coeffs*, *dsc_v_split*, *p_rated*, *v_max*, *f_downscale_threshold*)
Check if downscaling required, and apply it.

Returns (float) the factor

@see: Annex 1-7, p 68

`wltp.experiment.`**`calcEngineRevs_required`**(*V*, *gear_ratios*, *n_idle*, *v_stopped_threshold*)
Calculates the required engine-revolutions to achieve target-velocity for all gears.

Returns array: N_GEARS: a (#gears X #velocity) float-array, eg. [3, 150] –> gear(3), time(150)

Return type array: GEARS: a (#gears X #velocity) int-array, eg. [3, 150] –> gear(3), time(150)

@see: Annex 2-3.2, p 71

`wltp.experiment.`**`calcPower_available`**(*N_GEARS*, *n_idle*, *n_rated*, *p_rated*, *load_curve*, *p_safety_margin*)
@see: Annex 2-3.2, p 72

`wltp.experiment.`**`calcPower_required`**(*V*, *A*, *test_mass*, *f0*, *f1*, *f2*, *f_inertial*)
@see: Annex 2-3.1, p 71

`wltp.experiment.`**`decideClass`**(*class_limits*, *class3_velocity_split*, *mass*, *p_rated*, *v_max*)
@see: Annex 1, p 19

wltp.experiment.**downscaleCycle**(*V*, *f_downscale*, *phases*)
Downscale just by scaling the 2 phases demarked by the 3 time-points with different factors, no recursion as implied by the specs.

@see: Annex 1-7, p 64-68

wltp.experiment.**gearsregex**(*gearspattern*)

Parameters **gearspattern** – regular-expression or substitution that escapes decimal-bytes written as: `\g\d+` with adding +128, eg:

```
\g124|\g7 --> unicode(128+124=252)|unicode(128+7=135)
```

wltp.experiment.**possibleGears_byEngineRevs**(*V*, *A*, *N_GEARS*, *ngears*, *n_idle*, *n_min_drive*, *n_clutch_gear2*, *n_min_gear2*, *n_max*, *v_stopped_threshold*, *driveability_issues*)
Calculates the engine-revolutions limits for all gears and returns for which they are accepted.

My interpratation for Gear2 `n_min` limit:

```
                    _______________          _______________
                  ///INVALID///|   CLUTCHED   |  GEAR-2-OK
EngineRevs(N): 0-----------------------+---------------------------->
for Gear-2                     |       |      +--> n_clutch_gear2   := n_idle + MAX(
                               |       |                                0.15% * n_idle,
                               |       |                                3%    * n_range)
                               |       +---------> n_idle
                               +-----------------> n_min_gear2      := 90% * n_idle
```

Returns GEARS_YES: possibibilty for all the gears on each cycle-step (eg: [0, 10] == True –> gear(1) is possible for t=10)

Return type list(booleans, nGears x CycleSteps)

@see: Annex 2-3.2, p 71

wltp.experiment.**possibleGears_byPower**(*N_GEARS*, *P_REQ*, *n_idle*, *n_rated*, *p_rated*, *load_curve*, *p_safety_margin*, *driveability_issues*)
@see: Annex 2-3.1 & 3.3, p 71 & 72

wltp.experiment.**rule_a**(*bV*, *GEARS*, *CLUTCH*, *driveability_issues*, *re_zeros*)
Rule (a): Clutch & set to 1st-gear before accelerating from standstill.

Implemented with a regex, outside rules-loop: Also ensures gear-0 always followed by gear-1.

NOTE: Rule(A) not inside x2 loop, and last to run.

wltp.experiment.**rule_c2**(*bV*, *A*, *GEARS*, *CLUTCH*, *driveability_issues*, *re_zeros*)
Rule (c2): Skip 1st-gear while decelerating to standstill.

Implemented with a regex, outside rules-loop: Search for zeros in _reversed_ V & GEAR profiles, for as long Accel is negative. NOTE: Rule(c2) is the last rule to run.

wltp.experiment.**runCycle**(*V*, *A*, *P_REQ*, *gear_ratios*, *n_idle*, *n_min_drive*, *n_rated*, *p_rated*, *load_curve*, *params*)
Calculates gears, clutch and actual-velocity for the cycle (V). Initial calculations happen on engine_revs for all gears, for all time-steps of the cycle (N_GEARS array). Driveability-rules are applied afterwards on the selected gear-sequence, for all steps.

Parameters

- **V** – the cycle, the velocity profile

- **A** – acceleration of the cycle (diff over V) in m/sec^2

Returns CLUTCH: a (1 X #velocity) bool-array, eg. [3, 150] –> gear(3), time(150)

Return type array

wltp.experiment.**step_rule_b1**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule (b1): Do not skip gears while accelerating.

wltp.experiment.**step_rule_b2**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule (b2): Hold gears for at least 3sec when accelerating.

wltp.experiment.**step_rule_c1**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule (c1): Skip gears <3sec when decelerating.

wltp.experiment.**step_rule_d**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule (d): Cancel shifts after peak velocity.

wltp.experiment.**step_rule_e**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule (e): Cancel shifts lasting 5secs or less.

wltp.experiment.**step_rule_f**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule(f): Cancel 1sec downshifts (under certain circumstances).

wltp.experiment.**step_rule_g**(*t*, *pg*, *g*, *V*, *A*, *GEARS*, *driveability_issues*)
Rule(g): Cancel upshift during acceleration if later downshifted for at least 2sec.

3.2 Module: wltp.model

The hierarchical data and their schema for the WLTC calculator (defaults, WLTC-data) used by the Model and Experiments classes.

Attention: The documentation of this core module has several issues and needs work.

wltp.model.**merge**(*a*, *b*, *path*=[ ])
'merges b into a

wltp.model.**model_base**()
The base model for running a WLTC experiment.

It contains some default values for the experiment (ie the default 'full-load-curve' for the vehicles). But note that it this model is not valid - you need to iverride its attributes.

:return :json_tree: with the default values for the experiment.

wltp.model.**model_schema**()

Returns The json-schema(dict) for input/output of the WLTC experiment.

wltp.model.**wltc_data**()
The WLTC-data required to run an experiment (the class-cycles and their attributes)..

:return :json_tree:

wltp.model.**wltc_schema**()
The json-schema for the WLTC-data required to run a WLTC experiment.

:return :dict:

CHAPTER 4

Changes

Contents

4.1 Known deficiencies

- (!) Driveability-rules not ordered as defined in the latest task-force meeting.
- (!) The `n_min_drive` is not calculated as defined in the latest task-force meeting, along with other recent updates.
- (!) Does not discriminate between *Unladen mass* and *Test mass*.
- Clutching-points and therefore engine-speed are very preliminary.
- Results are not yet compared with the new wltp_db sample-vehicles.

4.2 TODOs

- Add cmd-line front-end.
- Add IPython front-end.
- No automatic calculation of masses and road-load coefficients from H & L vehicles.

4.3 Changelog

4.3.1 v0.0.8-alpha, 04-Aug-2014

- Documentation fixes.

4.3.2 v0.0.7-alpha, 31-Jul-2014: 1st *public*

- Rename project to 'wltp'.
- Switch license from AGPL –> EUPL (the same license assumed *retrospectively* for older version)
- Add wltp_db files.
- Unify instances & schemas in `model.py`.
- Possible to Build as standalone *.exe* using *cx_freeze*.
- **Preparations for PyPI/github distribution.**
  - Rename project to "wltp".
  - Prepare Sphinx documentation for http://readthedocs.org.
  - Update setup.py
  - Update project-coordinates (authors, etc)

4.3.3 v0.0.6-alpha, 5-Feb-2014

- Make it build as standalone *.exe* using *cx_freeze*.
- Possible to transplant base-gears and then apply on them driveability-rules.
- Embed Model –> Experiment to simplify client-code.
- Changes in the data-schema for facilitating conditional runs.
- More reverse-engineered comparisons with heinz's data.

4.3.4 v0.0.5-alpha, 18-Feb-2014

- Many driveability-improvements found by trial-n-error comparing with Heinz's.
- Changes in the data-schema for facilitating storing of tabular-data.
- Use Euro6 polynomial full_load_curve from Fontaras.
- Smooth-away INALID-GEARS.
- Make the plottings of comparisons of sample-vehicle with Heinz'results interactively report driveability-rules.
- Also report GEARS_ORIG, RPM_NORM, P_AVAIL, RPM, GEARS_ORIG, RPM_NORM results.

4.3.5 v0.0.4.alpha, 18-Jan-2014

- Starting to compare with Heinz's data - FOUND DISCREPANCIES IMPLTYING ERROR IN BASE CALCS.
- Test-enhancements and code for comparing with older runs to track algo behavior.
- Calc 'V_real'.
- Also report RPMS, P_REQ, DIRVEABILITY results.
- Make v_max optionally calculated from max_gear / gear_ratios.
- BUGFIX: in P_AVAIL 100% percents were mixed [0, 1] ratios!
- BUGFIX: make *goodVehicle* a function to avoid mutation side-effects.
- BUGFIX: add forgotten division on p_required Accel/3.6.
- BUGFIX: velocity-profile mistakenly rounded to integers!
- BUGFIX: v_max calculation based on n_rated (not 1.2 * n_rated).
- FIXME: get default_load_curve floats from Heinz-db.
- FIXME: what to to with INVALID-GEARS?

4.3.6 v0.0.3_alpha, 22-Jan-2014

- **-Driveability rules not-implemented:**
  - missing some conditions for rule-f.
  - no test-cases.
  - No velocity_real.
  - No preparation calculations (eg. vehicle test-mass).
  - Still unchecked for correctness of results.
- **-Pending Experiment tasks:**
  - FIXME: Apply rule(e) also for any initial/final gear (not just for i-1).
  - FIXME: move V–0 into own gear.
  - FIXME: move V–0 into own gear.
  - FIXME: NOVATIVE rule: "Clutching gear-2 only when Decelerating.".
  - FIXME: What to do if no gear foudn for the combination of Power/Revs??
  - NOTE: "interpratation" of specs for Gear-2
  - NOTE: Rule(A) not needed inside x2 loop.
  - NOTE: rule(b2): Applying it only on non-flats may leave gear for less than 3sec!
  - NOTE: Rule(c) should be the last rule to run, outside x2 loop.
  - NOTE: Rule(f): What if extra conditions unsatisfied? Allow shifting for 1 sec only??
  - TODO: Construct a matrix of n_min_drive for all gears, including exceptions for gears 1 & 2.
  - TODO: Prepend row for idle-gear in N_GEARS
  - TODO: Rule(f) implement further constraints.

  - TODO: Simplify V_real calc by avoiding multiply all.

4.3.7 v0.0.2_alpha, 7-Jan-2014

- -Still unchecked for correctness of results.

4.3.8 v0.0.1, 6-Jan-2014: Alpha release

- -Unchecked for correctness.
- Runs OK.
- Project with python-packages and test-cases.
- Tidied code.
- Selects appropriate classes.
- Detects and applies downscale.
- Interpreted and implemented the nonsensical specs concerning `n_min` engine-revolutions for gear-2 (Annex 2-3.2, p71).
- -Not implemented yet driveability rules.
- -Does not output real_velocity yet - inly gears.

4.3.9 v0.0.0, 11-Dec-2013: Inception stage

- Mostly setup.py work, README and help.

CHAPTER 5

Indices

5.1 Glossary

WLTP The Worldwide harmonised Light duty vehicles Test Procedure, a *GRPE* informal working group

UNECE The United Nations Economic Commission for Europe, which has assumed the steering role on the *WLTP*.

GRPE UNECE Working party on Pollution and Energy – Transport Programme

GTR Global Technical Regulation

WLTC The family of the 3 pre-defined *driving-cycles* to use for each vehicle depending on its *PMR*. Classes 1,2 & 3 are split in 2, 4 and 4 *parts* respectively.

PMR The `rated_power / unladen_mass` of the vehicle

Unladen mass *UM* or *Curb weight*, the weight of the vehicle in running order minus the mass of the driver.

Test mass *TM*, the representative weight of the vehicle used as input for the calculations of the simulation, derived by interpolating between high and low values for the CO_2-family of the vehicle.

Downscaling Reduction of the top-velocity of the original drive trace to be followed, to ensure that the vehicle is not driven in an unduly high proportion of "full throttle".

5.2 *Index*

CHAPTER 6

Glossary

WLTP The Worldwide harmonised Light duty vehicles Test Procedure, a *GRPE* informal working group

UNECE The United Nations Economic Commission for Europe, which has assumed the steering role on the *WLTP*.

GRPE UNECE Working party on Pollution and Energy – Transport Programme

GTR Global Technical Regulation

WLTC The family of the 3 pre-defined *driving-cycles* to use for each vehicle depending on its *PMR*. Classes 1,2 & 3 are split in 2, 4 and 4 *parts* respectively.

PMR The `rated_power / unladen_mass` of the vehicle

Unladen mass *UM* or *Curb weight*, the weight of the vehicle in running order minus the mass of the driver.

Test mass *TM*, the representative weight of the vehicle used as input for the calculations of the simulation, derived by interpolating between high and low values for the CO_2-family of the vehicle.

Downscaling Reduction of the top-velocity of the original drive trace to be followed, to ensure that the vehicle is not driven in an unduly high proportion of "full throttle".

W

Symbols

A

C

D

E

G

M

P

R

S

T

U

W