wltp Documentation

Release 0.0.9-alpha.4

Authors: see '4.5 Development Team' section

January 22, 2015

Contents

Release 0.0.9-alpha.4

Documentation https://wltp.readthedocs.org/

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

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

Keywords UNECE, automotive, car, cars, driving, engine, fuel-consumption, gears, gearshifs, rpm, simulation, simulator, standard, vehicle, vehicles, wltc

The *wltp* is a python package that calculates the *gear-shifts* of Light-duty vehicles running the *WLTP* driving-cycles, according to *UNECE*'s GTR (Global Technical Regulation) draft.

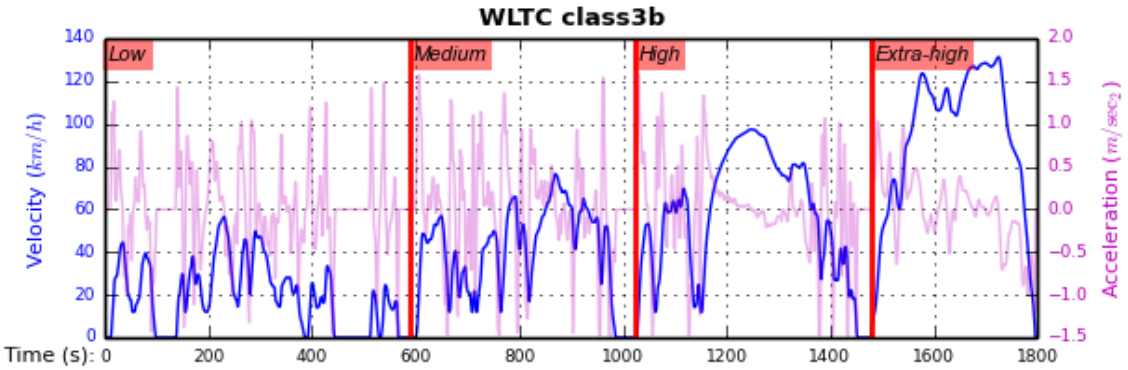


Figure 1: **Figure 1:** *WLTP cycle for class-3b Vehicles*

Attention: This *wltp* python project is still in *alpha* stage. Its results are not considered "correct", and no experimental procedures should rely currently on them.
Some of the known deficiencies are described in these places:

- In the *Changes*.
- Presented in the diagrams of the *Tests, Metrics & Reports* section.
- Imprinted in the `wltp_db_tests` test-case (automatically comparared with a pre-determined set of vehicles from Heinz-db on each build) Currently, mean rpm differ from Heinz-db < 0.5% and gears diff < 5% for a 1800-step class-3 cycle.

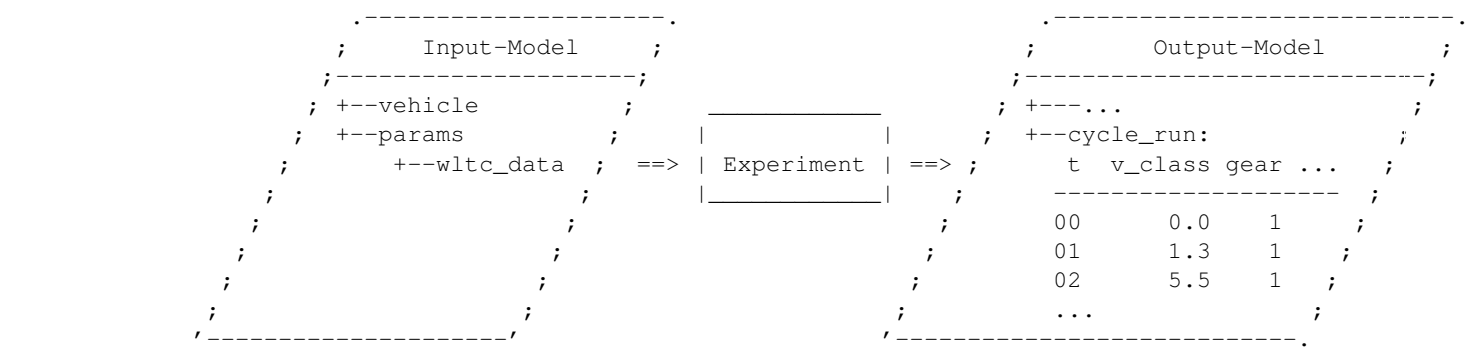
CHAPTER 1

Introduction

1.1 Overview

The calculator accepts as input the vehicle's technical data, along with parameters for modifying the execution of the *WLTC* cycle, and it then spits-out the gear-shifts of the vehicle, the attained speed-profile, and any warnings. It does not calculate any CO_2 emissions.

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



The *Input & Output Data* are instances of *pandas-model*, trees of strings and numbers, assembled with:

- sequences,
- dictionaries,
- `pandas.DataFrame`,
- `pandas.Series`, and
- URI-references to other model-trees.

1.2 Quick-start

On *Windows*/*OS X*, it is recommended to use one of the following "scientific" python-distributions, as they already include the native libraries and can install without administrative priviledges:

- WinPython (*Windows* only),
- Anaconda,
- Canopy,

Assuming you have a working python-environment, open a *command-shell*, (in *Windows* use **cmd.exe** BUT ensure **python.exe** is in its `PATH`), you can try the following commands:

Install

```
$ pip install wltp                          ## Use `--pre` if version-string has a build-suf
$ wltp --winmenus                           ## Adds StartMenu-items, Windows only.
```

See: *Install*

Cmd-line

```
$ wltp --version
0.0.9-alpha.4

$ wltp --help
...
```

See: *Cmd-line usage*

GUI

```
$ wltp --gui`                               ## For exploring model, but not ready yet.
```

Excel

```
$ wltp --excelrun                           ## Windows & OS X only
```

See: *Excel usage*

Python-code

```
from wltp.experiment import Experiment

input_model = { ... }           ## See also "Python Usage" for model contents.
exp = Experiment(input_model)
output_model = exp.run()
print('Results: \n%s' % output_model['cycle_run'])
```

See: *Python usage*

Tip: The commands beginning with `$`, above, imply a *Unix* like operating system with a *POSIX* shell (*Linux*, *OS X*). Although the commands are simple and easy to translate in its *Windows* counterparts, it would be worthwile to install Cygwin to get the same environment on *Windows*. If you choose to do that, include also the following packages in the *Cygwin*'s installation wizard:

```
* git, git-completion
* make, zip, unzip, bzip2
* openssh, curl, wget
```

But do not install/rely on cygwin's outdated python environment.

Tip: To install *python*, you can try the free (as in beer) distribution Anaconda for *Windows* and *OS X*, or the totally free WinPython distribution, but only for *Windows*:

- For *Anaconda* you may need to install project's dependencies manually (see `setup.py`) using **conda**.
- The most recent version of *WinPython* (python-3.4) although it has just changed maintainer, it remains a higly active project, and it can even compile native libraries using an installations of *Visual Studio*, if available (required for instance when upgrading `numpy/scipy`, `pandas` or `matplotlib` with **pip**).

  You must also **Register your WinPython installation** and **add your installation into** `PATH` (see *FAQ*). To register it, go to *Start menu* → *All Programs* → *WinPython* → *WinPython ControlPanel*, and then *Options* → *Register Distribution* .

See *Install* for more details

1.3 Discussion

CHAPTER 2

Install

Current 0.0.9-alpha.4 runs on Python-2.7+ and Python-3.3+ but 3.3+ is the preferred one, i.e, the desktop UI runs only with it. It is distributed on Wheels.

Before installing it, make sure that there are no older versions left over. So run this command until you cannot find any project installed:

```
$ pip uninstall wltp                          ## Use 'pip3' if both python-2 & 3 are in PATH.
```

You can install the project directly from the *PyPi* repo the "standard" way, by typing the **pip** in the console:

```
$ pip install wltp                            ## Use '--pre' if version-string has a build-suffix.
```

- If you want to install a *pre-release* version (the version-string is not plain numbers, but ends with `alpha`, `beta.2` or something else), use additionally `--pre`.
- If you want to upgrade an existing instalation along with all its dependencies, add also `--upgrade` (or `-U` equivalently), but then the build might take some considerable time to finish. Also there is the possibility the upgraded libraries might break existing programs(!) so use it with caution, or from within a *virtualenv* (isolated Python environment).
- To install it for different Python environments, repeat the procedure using the appropriate **python.exe** interpreter for each environment.
- ---

  Tip: To debug installation problems, you can export a non-empty `DISTUTILS_DEBUG` and *distutils* will print detailed information about what it is doing and/or print the whole command line when an external program (like a C compiler) fails.

After installation, it is important that you check which version is visible in your `PATH`:

```
$ wltp --version
0.0.9-alpha.4
```

To install for different Python versions, repeat the procedure for every required version.

2.1 Older versions

An additional purpose of the versioning schema of the project is to track which specific version of the GTR it implements. Given a version number `MAJOR.MINOR.PATCH`, the `MAJOR` part tracks the GTR phase implemented. See the "GTR version matrix" section in *Changes* for the mapping of MAJOR-numbers to GTR versions.

To install an older version issue the console command:

```
$ pip install wltp=1.1.1                      ## Use '--pre' if version-string has a build-suffix.
```

If you have another version already installed, you have to use `--ignore-installed` (or `-I`). For using the specific version, check this (untested) stackoverflow question .

Of course it is better to install each version in a separate *virtualenv* (isolated Python environment) and shy away from all this.

2.2 Installing from sources

If you download the sources you have more options for installation. There are various methods to get hold of them:

- Download the *source* distribution from *PyPi* repo.
- Download a release-snapshot from github
- Clone the *git-repository* at *github*.

  Assuming you have a working installation of git you can fetch and install the latest version of the project with the following series of commands:

  ```
  $ git clone "https://github.com/ankostis/wltp.git" wltp.git
  $ cd wltp.git
  $ python setup.py install                                    ## Use 'python3' if both python-2 &
  ```

When working with sources, you need to have installed all libraries that the project depends on:

```
$ pip install -r requirements/execution.txt .
```

The previous command installs a "snapshot" of the project as it is found in the sources. If you wish to link the project's sources with your python environment, install the project in development mode:

```
$ python setup.py develop
```

Note: This last command installs any missing dependencies inside the project-folder.

2.3 Project files and folders

The files and folders of the project are listed below:

```
+--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 and their schema for the calculation
|   +--experiment    ## (module) The calculator
|   +--plots         ## (module) Diagram-plotting code and utilities
+--docs/             ## Documentation folder
|   +--pyplots/      ## (scripts) Plot the metric diagrams embeded in the README
+--devtools/         ## (scripts) Preprocessing of WLTC data on GTR and the wltp_db
|   +--run_tests.sh ## (script) Executes all TestCases
+--wltp              ## (script) The cmd-line entry-point script for the calculator
+--setup.py          ## (script) The entry point for 'setuptools', installing, testing, etc
+--requirements/    ## (txt-files) Various pip-dependencies for tools.
+--README.rst
+--CHANGES.rst
+--LICENSE.txt
```

2.4 Discussion

CHAPTER 3

Usage

3.1 Cmd-line usage

Warning: Not implemented in yet.

The command-line usage below requires the Python environment to be installed, and provides for executing an experiment directly from the OS's shell (i.e. **cmd** in windows or **bash** in POSIX), and in a *single* command. To have precise control over the inputs and outputs (i.e. experiments in a "batch" and/or in a design of experiments) you have to run the experiments using the API python, as explained below.

The entry-point script is called **wltp**, and it must have been placed in your `PATH` during installation. This script can construct a *model* by reading input-data from multiple files and/or overriding specific single-value items. Conversely, it can output multiple parts of the resulting-model into files.

To get help for this script, use the following commands:

```
$ wltp --help                              ## to get generic help for cmd-line syntax
$ wltcmdp.py -M vehicle/full_load_curve    ## 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:

```
$ 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
```

3.2 GUI usage

Attention: Desktop UI requires Python 3!

For a quick-'n-dirty method to explore the structure of the model-tree and run an experiment, just run:

```
$ wltp --gui
```

3.3 Excel usage

Attention: Excel-integration requires Python 3 and *Windows* or *OS X*!

In *Windows* and *OS X* you may utilize the excellent xlwings library to use Excel files for providing input and output to the experiment.

To create the necessary template-files in your current-directory you should enter:

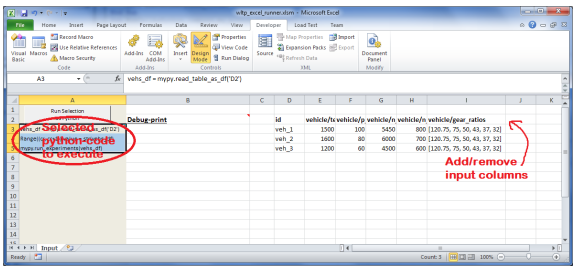
```
$ wltp --excel
```

You could type instead `wltp --excel` *`file_path`* to specify a different destination path.

In *windows*/*OS X* you can type `wltp --excelrun` and the files will be created in your home-directory and the excel will open them in one-shot.

All the above commands creates two files:

`wltp_excel_runner.xlsm` The python-enabled excel-file where input and output data are written, as seen in the screenshot below:



After opening it the first tie, enable the macros on the workbook, select the python-code at the left and click the *Run Selection as Pyhon* button; one sheet per vehicle should be created.

The excel-file contains additionally appropriate *VBA* modules allowing you to invoke *Python code* present in *selected cells* with a click of a button, and python-functions declared in the python-script, below, using the `mypy` namespace.

To add more input-columns, you need to set as column *Headers* the *json-pointers* path of the desired model item (see *Python usage* below,).

`wltp_excel_runner.py` Utility python functions used by the above xls-file for running a batch of experiments.

The particular functions included reads multiple vehicles from the input table with various vehicle characteristics and/or experiment parameters, and then it adds a new worksheet containing the cycle-run of each vehicle . Of course you can edit it to further fit your needs.

Note: You may reverse the procedure described above and run the python-script instead. The script will open the excel-file, run the experiments and add the new sheets, but in case any errors occur, this time you can debug them, if you had executed the script through *LiClipse*, or *IPython*!

Some general notes regarding the python-code from excel-cells:

- On each invocation, the predefined VBA module `pandalon` executes a dynamically generated python-script file in the same folder where the excel-file resides, which, among others, imports the "sister" python-script file. You can read & modify the sister python-script to import libraries such as 'numpy' and 'pandas', or pre-define utility python functions.
- The name of the sister python-script is automatically calculated from the name of the Excel-file, and it must be valid as a python module-name. Therefore do not use non-alphanumeric characters such as spaces('), `dashes` (-) and `dots` (.') on the Excel-file.
- On errors, a log-file is written in the same folder where the excel-file resides, for as long as **the message-box is visible, and it is deleted automatically after you click 'ok'!**
- Read http://docs.xlwings.org/quickstart.html

3.4 Python usage

Example python REPL (Read-Eval-Print Loop) example-commands are given below that setup and run an *experiment*.

First run **python** or **ipython** and try to import the project to check its version:

```
>>> import wltp

>>> wltp.__version__             ## Check version once more.
'0.0.9-alpha.4'

>>> wltp.__file__                ## To check where it was installed.
/usr/local/lib/site-package/wltp-...
```

If everything works, create the *pandas-model* that will hold the input-data (strings and numbers) of the experiment. You can assemble the model-tree by the use of:

- sequences,
- dictionaries,
- `pandas.DataFrame`,
- `pandas.Series`, and
- URI-references to other model-trees.

For instance:

```
>>> from wltp import model
>>> from wltp.experiment import Experiment
>>> from collections import OrderedDict as odic          ## It is handy to preserve keys-order.

>>> mdl = odic(
...   vehicle = odic(
...     unladen_mass = 1430,
...     test_mass    = 1500,
...     v_max        = 195,
...     p_rated      = 100,
...     n_rated      = 5450,
...     n_idle       = 950,
...     n_min        = None,                              ## Manufacturers my overridde it
...     gear_ratios         = [120.5, 75, 50, 43, 37, 32],
...     resistance_coeffs   = [100, 0.5, 0.04],
...   )
... )
```

For information on the accepted model-data, check its *JSON-schema*:

```
>>> model.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": [
...
```

You then have to feed this model-tree to the `Experiment` constructor. Internally the `Pandel` resolves URIs, fills-in default values and validates the data based on the project's pre-defined JSON-schema:

```
>>> processor = Experiment(mdl)          ## Fills-in defaults and Validates model.
```

Assuming validation passes without errors, you can now inspect the defaulted-model before running the experiment:

```
>>> mdl = processor.model                ## Returns the validated model with filled-in defaults.
>>> sorted(mdl)                          ## The "defaulted" model now includes the 'params' branch
['params', 'vehicle']
>>> 'full_load_curve' in mdl['vehicle'] ## A default wot was also provided in the 'vehicle'.
True
```

Now you can run the experiment:

```
>>> mdl = processor.run()                ## Runs experiment and augments the model with results.
>>> sorted(mdl)                          ## Print the top-branches of the "augmented" model.
['cycle_run', 'params', 'vehicle']
```

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.index.name = 't'
>>> df.shape                             ## ROWS(time-steps) X COLUMNS.
(1801, 11)
>>> df.columns
Index(['v_class', 'v_target', 'clutch', 'gears_orig', 'gears', 'v_real', 'p_available', 'p_require
>>> 'Mean engine_speed: %s' % df.rpm.mean()
'Mean engine_speed: 1940.72109939'
>>> df.describe()
           v_class     v_target     clutch   gears_orig        gears  \
count  1801.000000  1801.000000       1801  1801.000000  1801.000000
mean     46.506718    46.506718  0.0660744     3.794003     3.683509
std      36.119280    36.119280  0.2484811     2.278959     2.278108
...

            v_real  p_available   p_required          rpm     rpm_norm
count  1801.000000  1801.000000  1801.000000  1801.000000  1801.000000
mean     50.356222    28.846639     4.991915  1940.721099     0.214898
std      32.336908    15.833262    12.139823   840.959339     0.195142
...

>>> 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?
...
```

You can export the cycle-run results in a CSV-file with the following pandas command:

```
>>> df.to_csv('cycle_run.csv')
```

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

3.5 IPython notebook usage

The list of *IPython notebooks* for wltp is maintained at the wiki of the project.

3.5.1 Requirements

In order to run them interactively, ensure that the following requirements are satisfied:

1. A ipython-notebook server >= v2.x.x is installed for *python-3*, it is up, and running.
2. The *wltp* is installed on your system (see *Install* above).

3.5.2 Instructions

- Visit each *notebook* from the wiki-list that you wish to run and **download** it as `ipynb` file from the menu (*File|Download as...|IPython Notebook(.ipynb)*).
- Locate the downloaded file with your *file-browser* and **drag n' drop** it on the landing page of your notebook's server (the one with the folder-list).

Enjoy!

3.6 Discussion

CHAPTER 4

Getting Involved

This project is hosted in **github**. To provide feedback about bugs and errors or questions and requests for enhancements, use github's Issue-tracker.

4.1 Sources & Dependencies

To get involved with development, you need a POSIX environment to fully build it (*Linux*, *OSX* or *Cygwin* on *Windows*).

First you need to download the latest sources:

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

Virtualenv

You may choose to work in a *virtualenv* (isolated Python environment), to install dependency libraries isolated from system's ones, and/or without *admin-rights* (this is recommended for *Linux*/*Mac OS*).

> **Attention:** If you decide to reuse stystem-installed packages using `--system-site-packages` with `virtualenv <= 1.11.6` (to avoid, for instance, having to reinstall *numpy* and *pandas* that require native-libraries) you may be bitten by bug #461 which prevents you from upgrading any of the pre-installed packages with **pip**.

Liclipse IDE

Within the sources there are two sample files for the comprehensive LiClipse IDE:

- `eclipse.project`
- `eclipse.pydevproject`

Remove the `eclipse` prefix, (but leave the dot()) and import it as "existing project" from Eclipse's `File` menu.

Another issue is caused due to the fact that LiClipse contains its own implementation of *Git*, *EGit*, which badly interacts with unix *symbolic-links*, such as the `docs/docs`, and it detects working-directory changes even after a fresh checkout. To workaround this, Right-click on the above file *Properties* → *Team* → *Advanced* → *Assume Unchanged*

Then you can install all project's dependencies in *'development mode* using the `setup.py` script:

```
$ python setup.py --help                          ## Get help for this script.
Common commands: (see '--help-commands' for more)

  setup.py build      will build the package underneath 'build/'
```

```
  setup.py install    will install the package

Global options:
  --verbose (-v)      run verbosely (default)
  --quiet (-q)        run quietly (turns verbosity off)
  --dry-run (-n)      don't actually do anything
...

$ python setup.py develop                          ## Also installs dependencies into project's
$ python setup.py build                            ## Check that the project indeed builds ok.
```

You should now run the test-cases (see *Tests, Metrics & Reports*) to check that the sources are in good shape:

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

Note: The above commands installed the dependencies inside the project folder and for the *virtual-environment*. That is why all build and testing actions have to go through `python setup.py some_cmd`.

If you are dealing with installation problems and/or you want to permantly install dependant packages, you have to *deactivate* the virtual-environment and start installing them into your *base* python environment:

```
$ deactivate
$ python setup.py develop
```

or even try the more *permanent* installation-mode:

```
$ python setup.py install               # May require admin-rights
```

4.2 Development procedure

For submitting code, use `UTF-8` everywhere, unix-eol(`LF`) and set `git --config core.autocrlf = input`.

The typical development procedure is like this:

1. Modify the sources in small, isolated and well-defined changes, i.e. adding a single feature, or fixing a specific bug.
2. Add test-cases "proving" your code.
3. Rerun all test-cases to ensure that you didn't break anything, and check their *coverage* remain above 80%:

   ```
   $ python setup.py nosetests --with-coverage --cover-package wltp.model,wltp.experiment --
   ```

   Tip: You can enter just: `python setup.py test_all` instead of the above cmd-line since it has been *aliased* in the `setup.cfg` file. Check this file for more example commands to use during development.

4. If you made a rather important modification, update also the *Changes* file and/or other documents (i.e. README.rst). To see the rendered results of the documents, issue the following commands and read the result html at `build/sphinx/html/index.html`:

   ```
   $ python setup.py build_sphinx                  # Builds html docs
   $ python setup.py build_sphinx -b doctest       # Checks if python-code embeded in commen
   ```

5. If there are no problems, commit your changes with a descriptive message.
6. Repeat this cycle for other bugs/enhancements.
7. When you are finished, push the changes upstream to *github* and make a *merge_request*. You can check whether your merge-request indeed passed the tests by checking its build-status on the integration-server's site (TravisCI).

Hint: Skim through the small IPython developer's documentantion on the matter: The perfect pull request

4.3 Specs & Algorithm

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

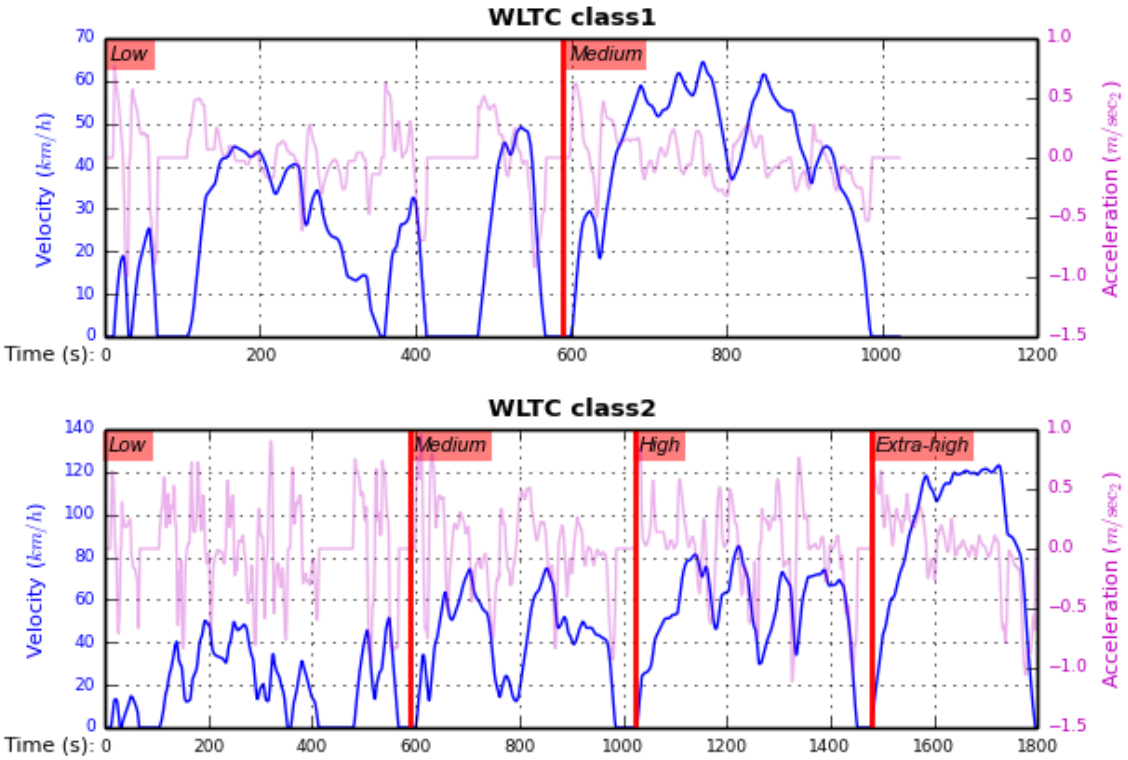
- http://www.unece.org/trans/main/wp29/wp29wgs/wp29grpe/grpedoc_2013.html
- https://www2.unece.org/wiki/pages/viewpage.action?pageId=2523179
- Probably a more comprehensible but older spec is this one: https://www2.unece.org/wiki/display/trans/DHC+draft+technical+

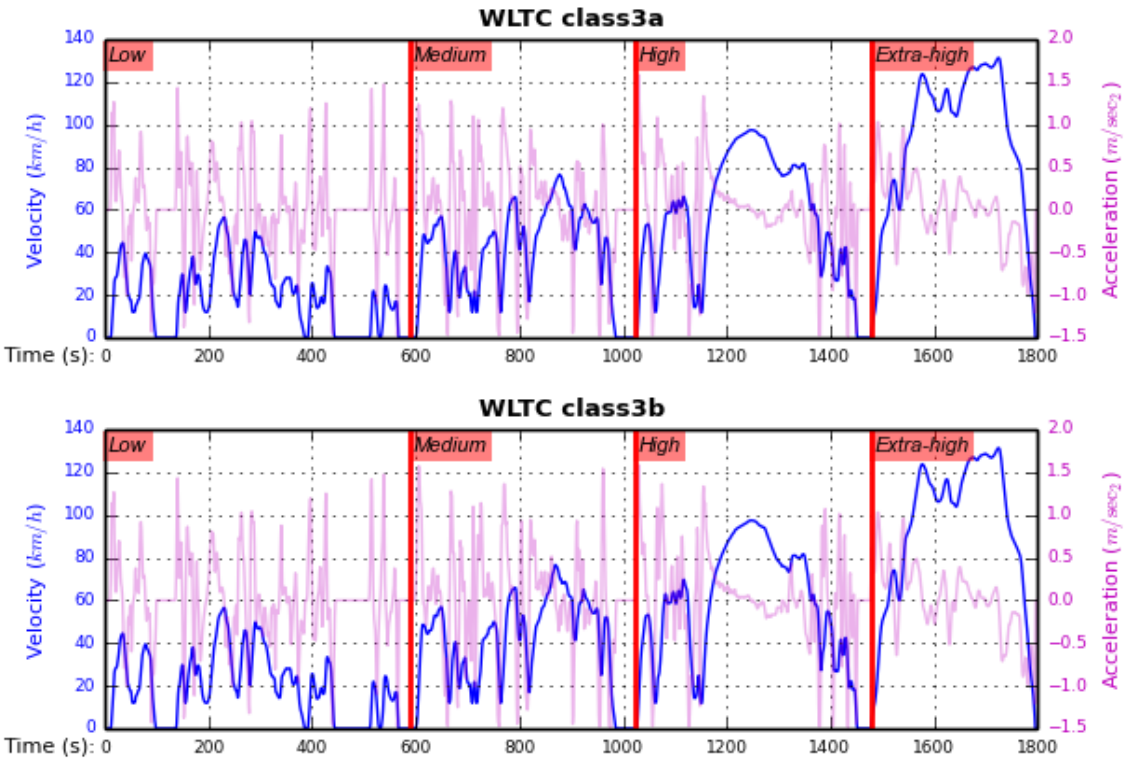
The WLTC-profiles for the various classes in the `devtools/data/cycles/` folder were generated from the tables of the specs above using the `devtools/csvcolumns8to2.py` script, but it still requires an intermediate manual step involving a spreadsheet to copy the table into ands save them as CSV.

Then use the `devtools/buildwltcclass.py` to construct the respective python-vars into the `wltp/model.py` sources.

Data-files generated from Steven Heinz's ms-access `vehicle info` db-table can be processed with the `devtools/preprocheinz.py` script.

4.3.1 Cycles





4.4 Development team

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

4.5 Discussion

CHAPTER 5

Tests, Metrics & Reports

In order to maintain the algorithm stable, a lot of effort has been put to setup a series of test-case and metrics to check the sanity of the results and to compare them with the Heinz-db tool or other datasets included in the project. These tests can be found in the `wltp/test/` folders.

Additionally, below are *auto-generated* representative diagrams with the purpose to track the behavior and the evolution of this project.

You can reuse the plotting code here for building nice ipython-notebooks reports, and (optionally) link them in the wiki of the project (see section above). The actual code for generating diagrams for these metrics is in `wltp.plots` and it is invoked by scripts in the `docs/pyplot/` folder.

5.1 Comparisons with Heinz-tool

This section compares the results of this tool to the Heinz's Access DB.

5.1.1 *Mean Engine-speed* vs *PMR*

First the mean engine-speed of vehicles are compared with access-db tool, grouped by PMRs:

Both tools generate the same rough engine speeds. There is though a trend for this project to produce lower rpm's as the PMR of the vehicle increases. But it is difficult to tell what each vehicle does isolated.

The same information is presented again but now each vehicle difference is drawn with an arrow:

It can be seen now that this project's calculates lower engine-speeds for classes 1 & 3 but the trend is reversed for class 2.

5.1.2 *Mean Engine-speed* vs *Gears*

Below the mean-engine-speeds are drawn against the mean gear used, grouped by classes and class-parts (so that, for instance, a class3 vehicle corresponds to 3 points on the diagram):

5.2 Comparisons with Older versions

[TBD]

5.2.1 Discussion

CHAPTER 6

FAQ

6.1 General

6.1.1 Who is behind this? How to contact the authors?

The immediate involved persons is described in the *Development team* section. The author is a participating member in the *GS Task-Force* on behalf of the EU Commission (JRC).

To contact the authors, use the DISQUS conversation, below, or create an issue.

6.1.2 What is the status of the project? Is it "official"?

It is a work-in-progress. It is aimed to become "official".

6.1.3 What is the roadmap for this project?

- Short-term plans are described in the *TODOs* section of *Changes*.
- In the longer run, it is expected to incorporate more *WLTP* calculations and reference data so that this projects acts as repository for diagrams and technical reports on those algorithms.

6.1.4 Can I copy/extend it? What is its License, in practical terms?

In a broad view, the core algorithm of the project is "copylefted" with the *EUPL-1.1+ license*, and it includes files from other "non-copyleft" open source licenses like *MIT MIT License* and *Apache License*, appropriately marked as such. So in an nutshell, you can study it, copy it, modify or extend it, and distrbute it, as long as you always distribute the sources of your changes.

6.2 Technical

6.2.1 I followed the instructions but i still cannot install/run/do *X*. What now?

If you have no previous experience in python, setting up your environment and installing a new project is a demanding, but manageable, task. Here is a checklist of things that might go wrong:

- Did you send each command to the **appropriate shell/interpreter**?

  You should enter sample commands starting `$` into your *shell* (**cmd** or **bash**), and those starting with `>>>` into the *python-interpreter* (but don't include the previous symbols and/or the *output* of the commands).

- Is **python contained in your PATH** ?

  To check it, type `python` in your console/command-shell prompt and press `[Enter]`. If nothing happens, you have to inspect `PATH` and modify it accordingly to include your python-installation.

  - Under *Windows* type `path` in your command-shell prompt. To change it, run **regedit.exe** and modify (or add if not already there) the `PATH` string-value inside the following *registry-setting*:

    `HKEY_CURRENT_USER\Environment\`

    You need to logoff and logon to see the changes.

    Note that *WinPython* **does not modify your path!** if you have registed it, so you definetely have to perform the the above procedure yourself.

  - Under *Unix* type `echo $PATH$` in your console. To change it, modify your "rc' files, ie: `~/.bashrc` or `~/.profile`.

- Is the correct **version of python** running?

  Certain commands such as **pip** come in 2 different versions *python-2 & 3* (**pip2** and **pip3**, respectively). Most programs report their version-infos with `--version`. Use `--help` if this does not work.

- Have you **upgraded/downgraded the project** into a more recent/older version?

  This project is still in development, so the names of data and functions often differ from version to version. Check the *Changes* for point that you have to be aware of when upgrading.

- Did you search whether **a similar issue** has already been reported?

- Did you **ask google** for an answer??

- If the above suggestions still do not work, feel free to **open a new issue** and ask for help here. Write down your platform (Windows, OS X, Linux), your exact python distribution and version, and include the *print-out of the failed command along with its error-message.*

  This last step will improve the documentation and help others as well.

6.2.2 I do not have python / cannot install it. Is it possible to try a *demo*?

Create an account into Wakari and post a Disqus-comment below requesting JRC's shared IPython notebook.

6.3 Discussion

CHAPTER 7

API reference

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

```
pandel
model
experiment
idgears
```

Among the various tests, those running on 'sample' databases for comparing differences with existing tool are the following:

```
samples_db_tests
wltp_db_tests
```

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

- `devtools/preprocheinz.py`
- `devtools/printwltcclass.py`
- `devtools/csvcolumns8to2.py`

7.1 Module: `wltp.experiment`

7.2 Module: `wltp.model`

7.3 Module: `wltp.pandel`

7.4 Module: `wltp.test.samples_db_tests`

7.5 Module: `wltp.test.wltp_db_tests`

CHAPTER 8

Changes

Contents

8.1 GTR version matrix

Given a version number `MAJOR.MINOR.PATCH`, the `MAJOR` part tracks the GTR phase implemented. The following matrix shows these correspondences:

Release train	GTR ver
0.0.9	Un-precise GTR phase-1a (diffs explained below)
0.1.x	GTR phase-1a (TODO)
1.x.x	GTR phase 1b [TDB]

8.2 Known deficiencies

- (!) Driveability-rules not ordered as defined in the latest task-force meeting.
- (!) The driveability-rules when speeding down to a halt is broken, and human-drivers should improvise.
- (!) The `n_min_drive` is not calculated as defined in the latest task-force meeting, along with other recent updates.

- (!) The `n_max` is calculated for ALL GEARS, resulting in "clipped" velocity-profiles, leading to reduced rpm's for low-powered vehicles.
- Clutching-points and therefore engine-speed are very preliminary (ie `rpm` when starting from stop might be < `n_idle`).

8.3 TODOs

- Add cmd-line and UI front-ends.
- Automatically calculate masses from H & L vehicles, and regression-curves from categories.
- wltp_db: Improve test-metrics with group-by classes/phases.
- model: Enhance model-preprocessing by interleaving "octapus" merging stacked-models between validation stages.
- model: finalize data-schema (renaming columns and adding `name` fields in major blocks).
- model/core: Accept units on all quantities.
- core: Move calculations as class-methods to provide for overriding certain parts of the algorithm.
- core: Support to provide and override arbitrary model-data, and ask for arbitrary output-ones by topologically sorting the graphs of the calculation-dependencies.
- build: Separate wltpdb tests as a separate, optional, plugin of this project (~650Mb size).

8.4 Changelog

8.4.1 v0.0.9-alpha.4 (XX-Jan-2015)

Same algo as `alpha.3` but with corrected engine-speed for idle. It is used for reports and simulation run by JRC to build the CO_2MPAS model, but still not driveable due to downshifting to 1st-gear when stopping to standstill.

- core, model: Possible to define different `n_min_drive` & `f_safety_margins` per gear.
- core: Add function to identify gear-ratios from experimental engine-runs.
- excel, tests: Add ExcelRunner TCs.

8.4.2 v0.0.9-alpha.3 (1-Dec-2014)

This is practically the 1st public releases, reworked in many parts, much better documented, continuously tested and build using TravisCI, with on-the-fly generated diagrams as metrics, BUT the arithmetic results produced are still identical to v0.0.7, so that the test-cases and metrics still describe that version, for future comparison.

- Use CONDA for running on TravisCI.
- Improve ExcelRunner.
- docs and metrics improvements.
- ui: Added Excel frontend.
- ui: Added desktop-UI proof-of-concept (`wltp.tkui`).
- metrics: Add diagrams auto-generated from test-metrics into generated site (at "Getting Involved" section).

Noteworthy or *incompatilble* changes

- **Code:**
  - package `wltc` –> `wltp`
  - class `Experiment` –> `Processor`
- **Model changes:**
  - `/vehicle/mass` –> (`test_mass` and `unladen_mass`)
  - `/cycle_run`: If present, (some of) its columns override the calculation.
- Added Excel front-end.
- Added *Metrics* section in documents whith on-the-fly generated diagrams comparing and tracking the behavior of the algorithm.
- Now the Eclipse's PyDev-project files are included only as templates; copy them and remove the `eclipse` prefix before importing project into Eclipse/Liclipse.

8.4.3 v0.0.9-alpha.1 (1-Oct-2014)

- Backported also to Python-2.7.
- model, core: Discriminate between *Test mass* from *Unladen mass* (optionally auto-calced by `driver_mass` = 75(kg)).
- model, core: Calculate default resistance-coefficients from a regression-curve (the one found in Heinz-db).
- model, core: Possible to overide WLTP-Class, Target-V & Slope, Gears if present in the `cycle_run` table.
- model: Add NEDC cycle data, for facilitating comparisons.
- tests: Include sample-vehicles along with the distribution.
- tests: Speed-up tests by caching files to read and compare.
- docs: Considerable improvements, validate code in comments and docs with *doctest*.
- docs: Provide a http-link to the list of IPython front-ends in the project's wiki.
- build: Use TravisCI as integration server, Coveralls.io as test-coverage service-providers.
- build: Stopped .EXE distribution; need a proper python environment.

8.4.4 v0.0.8-alpha(04-Aug-2014), v0.0.8.alpha.2(1-Dec-2014)

- Documentation fixes.

8.4.5 v0.0.7-alpha, 31-Jul-2014: 1st *internal*

Although it has already been used in various exercises internally in JRC, it never graduated out of *Alpha* state.

- 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)

8.4.6 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.

8.4.7 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.

8.4.8 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?

8.4.9 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.

8.4.10 v0.0.2_alpha, 7-Jan-2014

- -Still unchecked for correctness of results.

8.4.11 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.

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

- Mostly setup.py work, README and help.

CHAPTER 9

Indices

9.1 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

GS Task-Force The Gear-shift Task-force of the *GRPE*. It is the team of automotive experts drafting the gear-shifting strategy for vehicles running the *WLTP* cycles.

WLTC The family of pre-defined *driving-cycles* corresponding to vehicles with different PMR (Power to Mass Ratio). Classes 1,2, 3a & 3b are split in 2, 4, 4 and 4 *parts* respectively.

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".

pandas-model The *container* of data that the gear-shift calculator consumes and produces. It is implemented by `wltp.pandel.Pandel` as a mergeable stack of *JSON-schema* abiding trees of strings and numbers, formed with sequences, dictionaries, `pandas`-instances and URI-references.

JSON-schema The JSON schema is an IETF draft that provides a *contract* for what JSON-data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. You can learn more about it from this excellent guide, and experiment with this on-line validator.

JSON-pointer JSON Pointer(**RFC 6901**) defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document. It aims to serve the same purpose as *XPath* from the XML world, but it is much simpler.

9.1.1 *Index*

CHAPTER 10

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

GS Task-Force The Gear-shift Task-force of the *GRPE*. It is the team of automotive experts drafting the gear-shifting strategy for vehicles running the *WLTP* cycles.

WLTC The family of pre-defined *driving-cycles* corresponding to vehicles with different PMR. Classes 1,2, 3a & 3b are split in 2, 4, 4 and 4 *parts* respectively.

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".

pandas-model The *container* of data that the gear-shift calculator consumes and produces. It is implemented by `wltp.pandel.Pandel` as a mergeable stack of *JSON-schema* abiding trees of strings and numbers, formed with sequences, dictionaries, `pandas`-instances and URI-references.

JSON-schema The JSON schema is an IETF draft that provides a *contract* for what JSON-data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. You can learn more about it from this excellent guide, and experiment with this on-line validator.

JSON-pointer JSON Pointer(**RFC 6901**) defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document. It aims to serve the same purpose as *XPath* from the XML world, but it is much simpler.

D

E

G

J

P

R

T

U

W