**OEDI Task 3 Documentation – Transient Data Generation**

*Transient Use Case 3.a – Data Conversion*

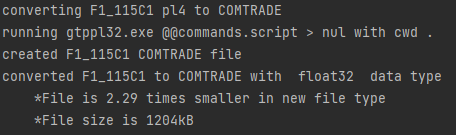
**Training Data Conversion Step**

After the transient simulations have been completed, the user has the option of converting the ATP output files in .pl4 format to point-on-wave COMTRADE files or compressed CSV via .mat file conversion. Both avenues for achieving the user preferred desired training data format are briefly discussed here. These instructions assume all .pl4 files reside in the local working directory.

1. COMTRADE file conversion:

Converting outputs to COMTRADE (Common Format for Transient Data Exchange) format yields a set of two files (.cfg and .dat) per .pl4 file converted using the built in GTPPLOT tool executable *GTPPL32.exe* included with the ATP installation. The .dat files will contain the transient waveform data and be much larger than the .cfg configuration files.

1. In the OEDI repository, access and place the files *pl42com.py* and *comtradeDT.py* in your local working directory.
2. In the OEDI repository, access and run the file *pl4\_to\_comtrade.py* from your local working directory. This file gathers all ATP output files in .pl4 format and converts to COMTRADE ASCII using the *GTPPL32.exe* toolbox (executable must be in local directory), then converts the file formatting to type float32. The pair of .cfg and .dat files dumped into your local directory will share similar labeling to the original .pl4. A successful conversion should print something similar to the following in the console run window:



1. MAT-to-Compressed CSV file conversion:

The conversion process from .pl4 files to .mat files can be accomplished traditionally by using MATLAB, however this process involves additional software requiring further licensing which may not be accessible to all users. Therefore, the method described in this section uses a combination of the built in ATP libraries and GNU Octave on Windows to maintain a pythonic conversion process using only open source software.

1. Install GNU Octave 8.2.0 (latest) – a free, open source, scientific programming language for handling mathematics-based tasks which offers “drop-in” compatibility with many MATLAB scripts. Octave runs on GNU/Linux, macOS and Windows (this application uses Windows), and can handle .mat files and .m scripts, offering familiarity to MATLAB users without the need for a license. Please download the latest version at [GNU Octave](https://octave.org/download) in the default location on your local machine.
2. After installation is complete, add the octave.exe file to your system environment variables and path (Windows) using C:\Program Files\GNU Octave\Octave-8.2.0\mingw64\bin. To be certain the installation is correct, open the command prompt and type ‘octave’ and press ENTER (Windows). If the program launches in the terminal, everything is correct. Else, check your installation location and path/env/system variables, and try again.
3. Install oct2py from instructions on [oct2py](https://pypi.org/project/oct2py/) in your python IDE (NumPy and SciPy must be already installed) and import. This library provides a unique bridge to run legacy .m files and handles MATLAB/OCTAVE style scripting in python.
4. In the OEDI repository, access and run the file *gen\_con\_matfiles.py* in your local working directory. This file gathers all ATP output files in .pl4 format, converts to .mat file version 4 using the *GTPPL32.exe* toolbox (executable must be in local directory), and upgrades to .mat version 6 using Octave. The resulting .mat files are stored in the local directory.
5. Finally, the files may be compressed to .hdf5 file format for training purposes and ease of data storage capacity at the users discretion. First, install hdf5storage using instruction from [hdf5](https://pythonhosted.org/hdf5storage/) in your python IDE and import. This package is similar to the h5py library but works well with established pythonic variables for fast compression storage. The file *gen\_con\_matfiles.py* is only set up to complete step IV, so the last function convertToHDF5() call must be UNCOMMENTED to add this step to the data conversion process (see line 100) by the user. Additional code at the end of the script may be used to read the hdf5 file back into your python script as a variable if necessary.

The next section of this document describes further data manipulation options for the user including various parameters and their details.

**Variables in Training Data**

After the transient data generation and conversion, the point-on-wave data can be used for any application. If the user in interested to use the data for fault identification and protection zone classification algorithm, then it’s essential to rearrange the data in a particular format along with the labels for supervised training. This section provides information about different variables that are available in the transient data file. When a user loads a data file (.pl4 or .mat or COMTRADE format), it contains

1. *vSBUS\_A, vSBUS\_B* and *vSBUS\_C* - substation point-on-wave 3phase voltage.
2. *iSBUS\_A25\_\_\_A, iSBUS\_A25\_\_\_B* and *iSBUS\_A25\_\_\_C -* substation point-on-wave 3phase current.
3. *iTACS\_\_PV001#, iTACS\_\_PV004#* and *iTACS\_\_PV014# -* number followed by *PV###* refers to the PV number. Here *PV014, PV004* and *PV001* are the highest capacity PVs on A, B and C phases respectively.
4. *iTACS\_\_PV001A, iTACS\_\_PV001X -* suffixes A, B, C specify the inverter's instantaneous phase voltages and suffixes X, Y, Z specify the inverter's instantaneous injected currents.
5. *iTACS\_\_PV001I, iTACS\_\_PV001V and iTACS\_\_PV001W* suffix V specifies the inverter's calculated RMS voltage, suffix *I* specifies the inverter's RMS current injection and suffix W specifies the inverter's estimated frequency (rad/s).