PNNL-26197

Dynamic Contingency Analysis Tool (DCAT) User Manual with Test System Examples

January 2017

NA Samaan
JE Dagle
YV Makarov
R Diao
B Vyakaranam
B Zhang
M Vallem
T Nguyen
FK Tuffner
J Conto
SW Kang

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <http://www.ntis.gov/about/form.aspx>
Online ordering: http://www.ntis.gov

This document was printed on recycled paper.
(8/2010)

PNNL-26197

Dynamic Contingency Analysis Tool (DCAT) User Manual with Test System Examples

NA Samaan
JE Dagle
YV Makarov
R Diao
B Vyakaranam
B Zhang[1]
M Vallem
T Nguyen
FK Tuffner
J Conto[2]
SW Kang[2]

January 2017

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

[1] Texas A&M University
[2] ERCOT

Summary

This document introduces the Dynamic Contingency Analysis Tool (DCAT) software package developed by the Pacific Northwest National Laboratory and serves as a guide for using this package to conduct cascading failure simulations. Additionally, the parallel processing module of the DCAT tool, which runs based on the “MPjobs” package developed by the Electric Reliability Council of Texas, is also introduced. Detailed instructions on how to use this module on multi-core Windows workstations or servers are also provided. Full explanation of the DCAT methodology is available in the Phase 1 report [1].

Section 1.0 lists the system requirements for running the DCAT tool. Sections 2.0, 3.0 and 4.0 describe how to use the DCAT package, as well as run the parallel processing of the DCAT process: section 2.0 introduces the key functions of different modules or subroutines, and the relationships among them; section 3.0 describes how to use the DCAT package to run different simulations; and section 4.0 shows how to use the “MPjobs” package to run parallel processing of the DCAT process.

The detailed introduction of the DCAT package is provided in sections 5.0 and 6.0. Section 5.0 introduces in detail the different modules in the DCAT package, and section 6.0 gives a detailed introduction to the module “MainDACT.py”, which is the main module to simulate the potential cascading events in a given contingency.

Acknowledgments

This project is funded by the U.S. Department of Energy Office of Electricity Delivery and Energy Reliability (DOE-OE). The project team wants to especially thank Mr. Gil Bindewald, Mr. Ali Ghassemian Program Managers and Dr. David Ortiz, Former Deputy Assistant Secretary for Energy Infrastructure Modeling and Analysis (OE-40) for their continuing support, help, and guidance.

The project team appreciates technical support in developing the Dynamic Contingency Analysis Tool from Siemens Power Technologies staff Mr. Hugo Raul Bashualdo, Mr. Dinemayer Silva, Mr. Jayapalan Senthil, Mr. James Feltes, Mr. Joseph Smith, and Mr. Krishnat Patil. We would like also to thank Siemens subcontractor Mr. Eli Pajuelo for his help in protection modeling in PSS/E.

The project team appreciates contributions of the following Pacific Northwest National Laboratory staff:

- Dr. Henry Huang, Analytics Subsector Manager
- Mr. Yousu Chen, Peer Reviewer
- Mr. Dale King, Project Management Office Director
- Mr. Carl Imhoff, Electricity Infrastructure Market Sector Manager
- Ms. Maura Zimmerschied, Technical Editor
- Ms. Carla Raymond, Project Specialist

Acronyms and Abbreviations

ACCC	AC contingency calculation
API	application program interface
DCAT	Dynamic Contingency Analysis Tool
IDE	integrated development environment
PSS®E	Siemens PTI PSS®E Power Flow software
SPS	special protection system

Contents

Figures

Tables

1.0 System Requirements

The system requirements for running the Dynamic Contingency Analysis Tool (DCAT) package are as follows:

- Windows 7 or later (64 bit)
- Siemens PSS®E 33.7
- Python 2.7
  - "NumPy":

    "NumPy" is a fundamental package for scientific computing with Python. Follow the download instructions on the NumPY web page: https://sourceforge.net/projects/numpy/files/NumPy/1.9.2/.
  - Eclipse or another integrated development environment (IDE) is highly recommended.
    - The recommended Eclipse packages are
      - Eclipse IDE for Eclipse Committers 4.5.2. Follow the download instructions on this version web page:

        http://www.eclipse.org/downloads/packages/eclipse-ide-eclipse-committers-452/mars2
      - Eclipse IDE for Java EEE Developers. Follow the download instructions on this version web page:

        http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2
  - Link Eclipse with PyDev to run Python script as PyDev project. PyDev requires Java 7 in order to run.

2.0 Introduction to the DCAT Package

The DCAT package, developed by Pacific Northwest National Laboratory (PNNL), consists of seven major Python modules and one configuration file, as listed in Table 2.1.

Table 2.1. Python Modules and Configuration File in the DCAT Package

File Name	Description
Supporingtools.py	This module defines some useful subroutines and serves as a supporting library to be used by other DCAT modules.
ReadLogFile.py	This module defines several useful subroutines for extracting tripping information from the PSS®E simulation log file, and serves as a support file to be called by other modules.
Make_DYR_withPNNLrelays_xx.py	This module creates ".sav" files and snapshots, ".snp" with converted models for the subsequent simulations. "_xx" corresponds to example number. For examples from 5 through 5 "_xx" should be replaced by "_35". For example 1 and 2, "_xx" should be replaced by "_01" and "_02", respectively.
RunFlatStart.py	This module runs a flat-start dynamic simulation using a given case.
RunTestFault.py	This module runs dynamic simulation of a given contingency with a given case.
MainDCAT.py	This module is the main function of the DCAT tool, and simulates the potential cascading events in a given contingency.
RunFlatStart_Config.ini	This configuration file sets up the parameters needed for the subsequent simulations. This file is only to run "RunFlatStart.py" and "RunTestFault.py".
set_DCAT_path.ini	This configuration file provides the necessary path linking to contingency definition settings and output folders for running "MainDCAT.py".

The relationship and the sequence of running the above modules are shown in

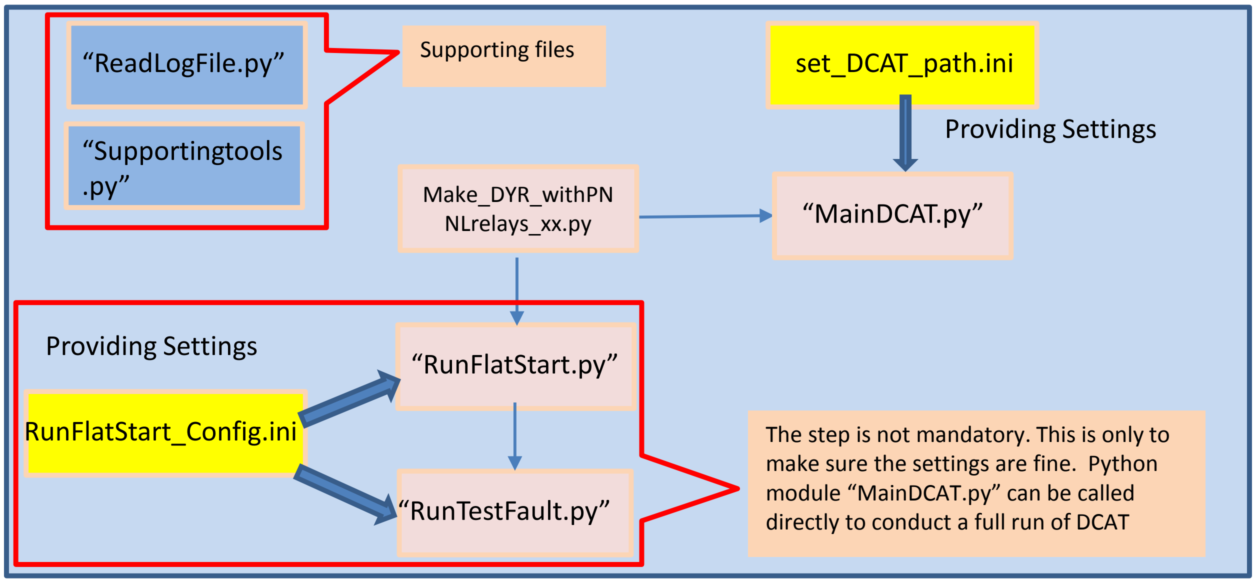

"ReadLogFile.py"
"Supportingtools .py"
Supporting files
set_DCAT_path.ini
Providing Settings
Make_DYR_withPN NLrelays_xx.py
"MainDCAT.py"
Providing Settings
RunFlatStart_Config.ini
"RunFlatStart.py"
"RunTestFault.py"
The step is not mandatory. This is only to make sure the settings are fine. Python module "MainDCAT.py" can be called directly to conduct a full run of DCAT

Figure 2.1. Relationship among Different Python Modules

The configuration file "RunFlatStart_Config.ini" provides the necessary settings for running the modules "RunFlatStart.py" and "RunTestFault.py". The module "RunFlatStart" is not mandatory. This step is designed to make sure settings in the model files are correct before running any contingencies or cascading failure analysis in DCAT. After the ".sav" and ".snp" files are generated, the users can choose either to run the dynamic simulation "RunTestFault" for a single contingency simulation or to run the full DCAT process for cascading failure analysis, in "MainDCAT".

3.0 How to Use the DCAT Package

3.1 Options for Running DCAT

3.1.1 Run Using Eclipse or Other IDE (recommended)

From our own experience, it is highly preferable to run DCAT from an IDE (Eclipse, for example).

Eclipse is an IDE that provides powerful code editing and debugging tools. Eclipse requires the PyDev extension to properly develop Python code.

3.1.1.1 Steps to Install PyDev

- From the menu of Eclipse, select “Help/Check for Updates” and install updates if necessary.
- Select “Help/Install New Software”.
- Enter http://pydev.sf.net/updates/ in the “Work with” box and click “Add…”
- Select “PyDev” and click “Next”.
- Follow directions and click “Finish”.
- Select “Window/Preferences”. In this window, select “PyDev/Interpreters” and choose “Python Interpreter”.
- At the top of the preferences pane, click “New...” and locate your installation of Python (python.exe)
- Click “Apply” and then “OK”.

3.1.1.2 Steps to Create a Python Project

- Click “File/New/PyDev project”.
- Enter the project name and click “Finish”.

3.1.1.3 Steps to Run Python within Eclipse

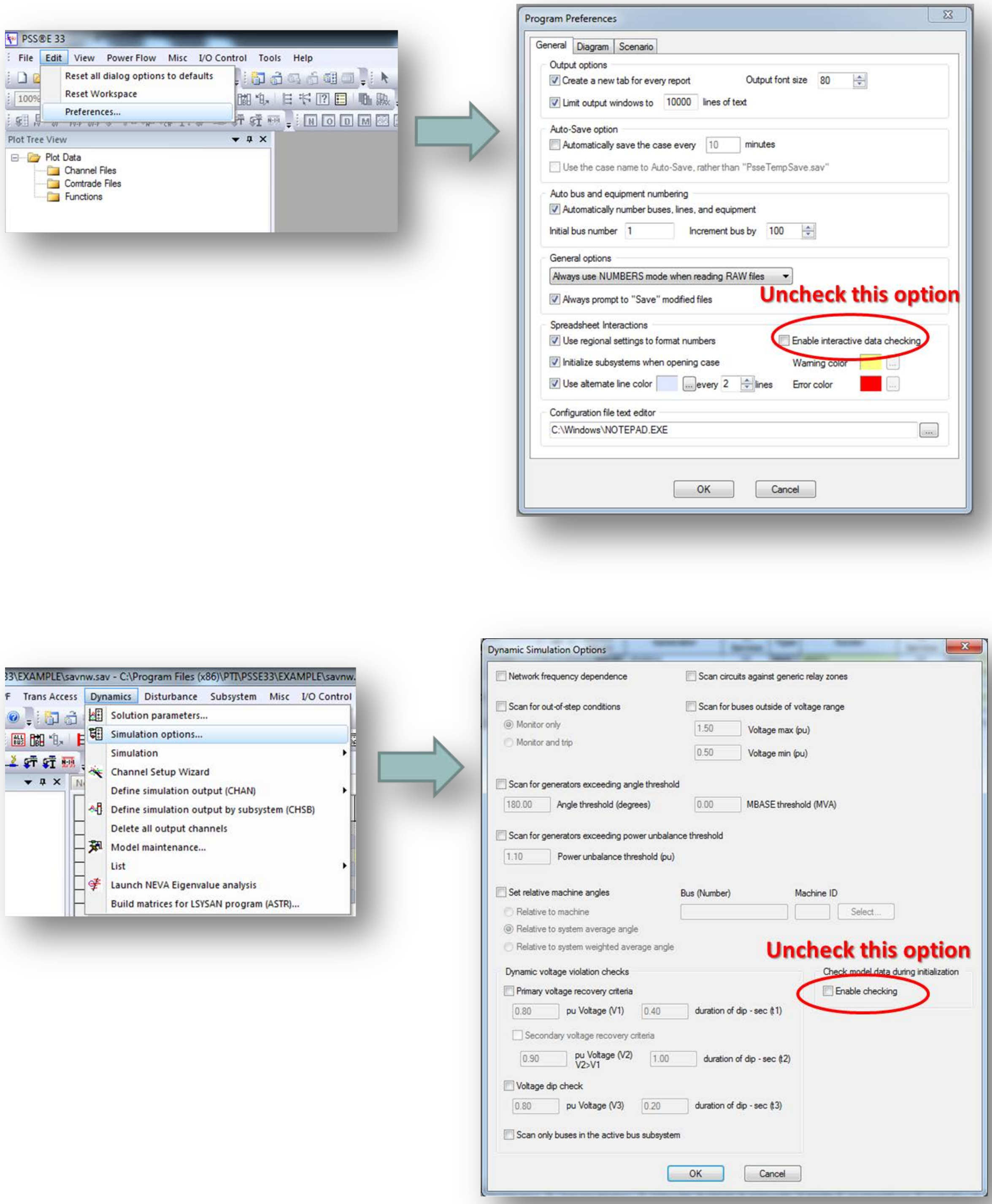
- Copy all Python and related files into the Python project folder.
- Right click on the Python file (.py file) you want to run.
- Chose “Run As/Python Run”. After you have run the code once, you can press the green “Run” arrow at the top of the Eclipse window to run it again.

3.1.2 Run Directly from PSS®E GUI

If the Python modules are being run directly in the PSS®E graphical user interface (GUI), two settings need to be modified:

1. In “Program Preferences”, uncheck “Enable interactive data checking” as shown in the upper panels of Figure 3.1.

2. In “Dynamic Simulation Options”, uncheck “Enable checking” as shown in the lower panels of Figure 3.1.



Program Preferences

General | Diagram | Scenario

Output options
- [✓] Create a new tab for every report — Output font size 80
- [✓] Limit output windows to 10000 lines of text

Auto-Save option
- [ ] Automatically save the case every 10 minutes
- [ ] Use the case name to Auto-Save, rather than "PsseTempSave.sav"

Auto bus and equipment numbering
- [✓] Automatically number buses, lines, and equipment
- Initial bus number 1 — Increment bus by 100

General options
- Always use NUMBERS mode when reading RAW files
- [✓] Always prompt to "Save" modified files

Spreadsheet Interactions
- [✓] Use regional settings to format numbers
- [ ] Enable interactive data checking
- [✓] Initialize subsystems when opening case — Warning color 🟨
- [✓] Use alternate line color every 2 lines — Error color 🟥

Configuration file text editor
- C:\Windows\NOTEPAD.EXE

Dynamic Simulation Options

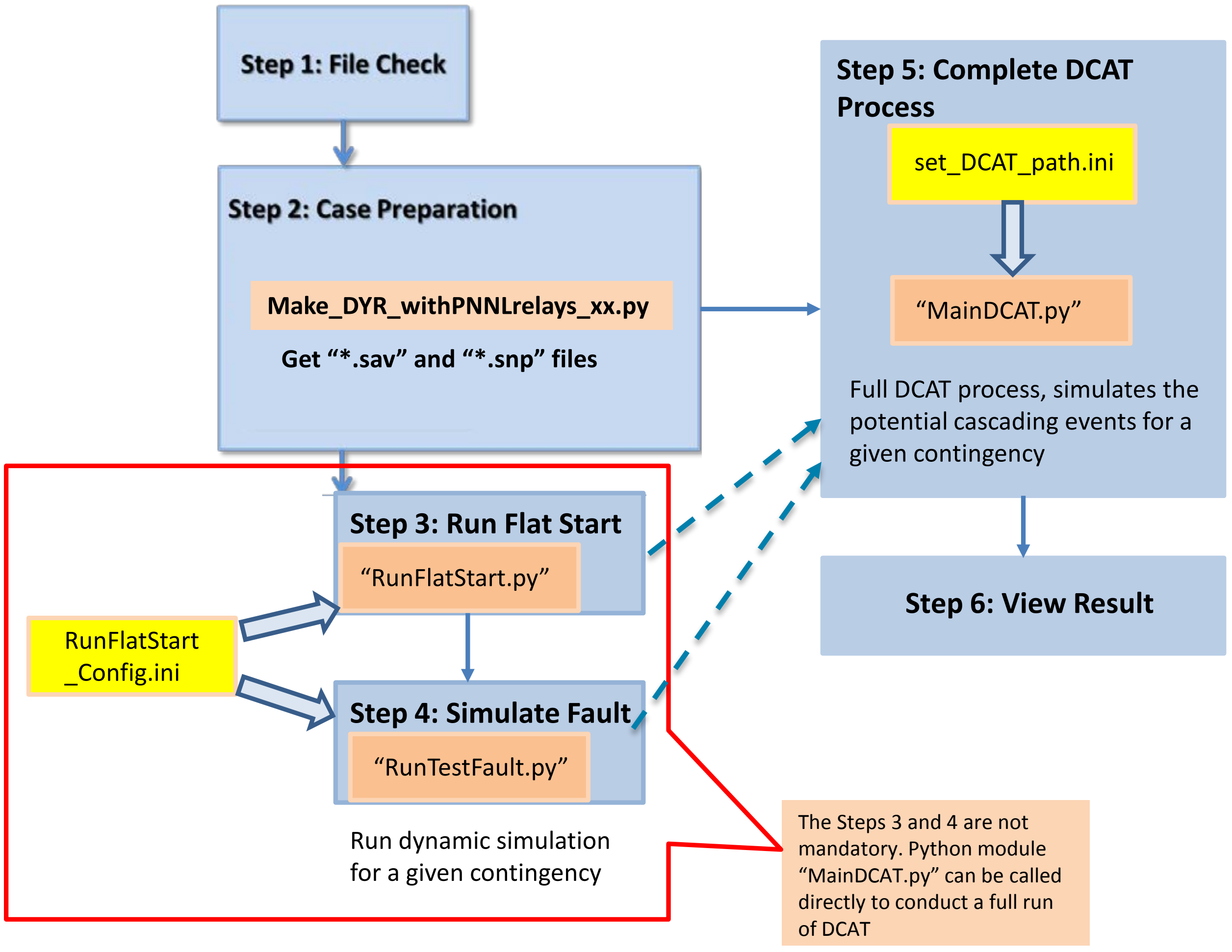
- [ ] Network frequency dependence
- [ ] Scan circuits against generic relay zones
- [ ] Scan for out-of-step conditions
  - (●) Monitor only
  - ( ) Monitor and trip
- [ ] Scan for buses outside of voltage range
  - 1.50 Voltage max (pu)
  - 0.50 Voltage min (pu)
- [ ] Scan for generators exceeding angle threshold
  - 180.00 Angle threshold (degrees)
  - 0.00 MBASE threshold (MVA)
- [ ] Scan for generators exceeding power unbalance threshold
  - 1.10 Power unbalance threshold (pu)
- [ ] Set relative machine angles — Bus (Number) — Machine ID — Select...
  - ( ) Relative to machine
  - (●) Relative to system average angle
  - ( ) Relative to system weighted average angle

Dynamic voltage violation checks
- [ ] Primary voltage recovery criteria
  - 0.80 pu Voltage (V1) — 0.40 duration of dip - sec (t1)
  - [ ] Secondary voltage recovery criteria
  - 0.90 pu Voltage (V2) V2>V1 — 1.00 duration of dip - sec (t2)
- [ ] Voltage dip check
  - 0.80 pu Voltage (V3) — 0.20 duration of dip - sec (t3)
- [ ] Scan only buses in the active bus subsystem

Check model data during initialization
- [ ] Enable checking

Figure 3.1. Selections for Running PSS®E Directly from The GUI

The following section describes how to use the DCAT package. There are five main steps to use the DCAT package, as illustrated in Figure 3.2. Sections 3.2 to 3.5 provide detailed instructions. Section 3.6 shows some examples and results.



Step 1: File Check
Step 2: Case Preparation
Make_DYR_withPNNLrelays_xx.py
Get "*.sav" and "*.snp" files
Step 5: Complete DCAT Process
set_DCAT_path.ini
"MainDCAT.py"
Full DCAT process, simulates the potential cascading events for a given contingency
Step 3: Run Flat Start
"RunFlatStart.py"
RunFlatStart_Config.ini
Step 4: Simulate Fault
"RunTestFault.py"
Run dynamic simulation for a given contingency
Step 6: View Result
The Steps 3 and 4 are not mandatory. Python module "MainDCAT.py" can be called directly to conduct a full run of DCAT

Figure 3.2. Main Steps for Running DCAT Package

Step 1: File Check

This step is to make sure that all files needed are ready and located in the correct folders. Section 3.2 provides more detailed instructions on this step.

Step 2: Case Preparation

This step is designed to prepare the converted ".sav" file and ".snp" file for the dynamic simulations. The Python module involved in this step is "Make_DYR_withPNNLrelays_xx.py". Section 3.3 provides more detailed instructions on this step.

Step 3: Run Flat Start

This step is not mandatory, and the purpose of this step is to make sure the case prepared in Step 1 is working well before any simulation is conducted. The Python module involved in this step is "RunFlatStart.py". Section 3.4 provides more detailed instruction on this step.

Step 4: Simulate Contingency

This step is to simulate the contingency in a dynamic simulation. The Python module “RunTestFault.py” will perform dynamic simulation for a specified contingency without cascading to other steps of DCAT.

Step 5: Complete DCAT Process

Python module “MainDCAT.py” is to conduct not only a single-stage dynamic simulation, but also the simulation of potential cascading events in a specified contingency. Section 3.5 provides more detailed instruction on this step.

Step 6: View Result

The main results include “.out” files generated during the dynamic simulation, “.log” files that record the process of the simulation, and “.csv” files that extract the main information during the simulation. The results will be saved in the folder “RESULTs” as default. (Users can change the path to save the output files through the configuration file).

3.2 File Check

Before starting to run any Python modules in the DCAT package, please make sure that the following files exist in the correct folder:

Under the main code folder “Main Data”

- The “.idv” file specifying the contingency
- The “.inl” file to be adopted in the inertia power flow
- The “.sub” file to be adopted in the alternating current (AC) contingency calculation (ACCC) process
- The “.mon” file to be adopted in the ACCC process

Under the subfolder “CASEs”

- The “.sav” file (serves as the base case file) that is going to be processed and converted in the “Make_DYR_withPNNLrelays_xx.py”

Under the subfolder “DYRs”

- The “.dyr” files (relay settings, etc.) that are going to be adopted in the “Make_DYR_withPNNLrelays_xx.py”; the details on how to set up the “.dyr” files can be found in Appendix A.

The files mentioned above are among the most important ones to be checked. It is recommended that other files that are to be adopted in the Python modules for other purposes be checked (especially when new files are to be adopted).

3.3 Case Preparation

3.3.1 Convert and Modify the Base Case

The Python module “Make_DYR_withPNNLrelays_xx.py” should be called to convert the power flow model and to modify any topology or network parameters. Please note the following:

1. The text parameter “logfile” in the file ‘config.ini’ sets the name of the “.log” file to be generated, and it can be modified according to users’ needs.
2. When loading the PSS®E case, make sure that the name of the “.sav” file is correct.
3. After calculating the power flow, the program then modifies the base case. Any modification to the base case should be made in the module “Make_DYR_withPNNLrelays_xx.py”. PNNL made some changes to the base case including the modification on the tap-ratio, rating of some transformers and the rate C of the branches. Those changes can be easily disabled by simply “commenting” them.
4. Finally, the converted case will be saved.

Note: the DCAT package has “Make_DYR_withPNNLrelays_01.py”, “Make_DYR_withPNNLrelays_02.py” and ““Make_DYR_withPNNLrelays_35.py” modules. The user has to choose and run the correct “**Make_DYR_withPNNLrelays _xx.py”** script as shown in Table 3.1.

Table 3.1. Make_DYR_withPNNLrelays _xx.py Options

File Name	Examples
Make_DYR_withPNNLrelays _35.py	3 through 5
Make_DYR_withPNNLrelays _01.py	1
Make_DYR_withPNNLrelays _02.py	2

3.3.2 Generate the “.snp” File

The Python module “**Make_DYR_withPNNLrelays** _xx.py” should be called to generate the “.snp” file. Note the following:

1. The text parameter “logfile” in the file ‘config.ini’ sets the name of the “.log” file to be generated, and it can be modified by users.
2. Make sure that the previously converted and saved “.sav” file is loaded here.
3. New “.dyr” files, if any, can be added in this Python module. If the new file is appended before “psspy.dyre_new”, a new compilation will be needed after this module is executed. Otherwise, there is no need to compile the file. It should be noted that psspy.dyre_new that is one of the APIs to clear dynamics working memory, read a Dynamics Data File, and place the model references specified on its data records into dynamics working memory (activity DYRE).
4. After adding the “.dyr” files, some modifications will be conducted. Any new modifications related to the dynamic data can be put here.
5. Finally, the “.snp” file will be saved, and the name of the file can be modified by users.

3.4 Run Flat Start

This step is not mandatory, as mentioned above. “RunFlatStart_Config.ini” and “RunFlatStart.py” are mainly involved in this step, and “RunFlatStart_Config.ini” provides the necessary settings for running “RunFlatStart.py”, which should be called to run a flat start to check whether the preparation of the case files is appropriate. The parameters necessary for running flat start in “RunFlatStart_Config.ini” are the following:

1. “logfile_flatstart”: this specifies the name of the “.log” file to be generated.
2. “savefile_flatstart”: this is the “.sav” case file for the simulation, and it should be the file generated in Section 3.3.1.
3. “snapfile_flatstart”: this is the “.snp” file for the simulation, and it should be the file generated in Section 3.3.2.
4. “outfile_flatstart”: this specifies the name of the “.out” file to be generated.
5. “Time_parameter_flatstart”: this determines the time duration of the dynamic simulation.
6. “timestep_flatstart”: this specifies the simulation time step for the flat start.
7. “filter_flatstart”: this specifies the filter time constant used in calculating bus frequency deviations for the flat start.

After finishing the parameter configuration, one can run “RunFlatStart.py” to run the flat start, and the “.log” file and the “.out” files, which will be generated in the main folder, can be checked to see whether the preparation of the case file is appropriate.

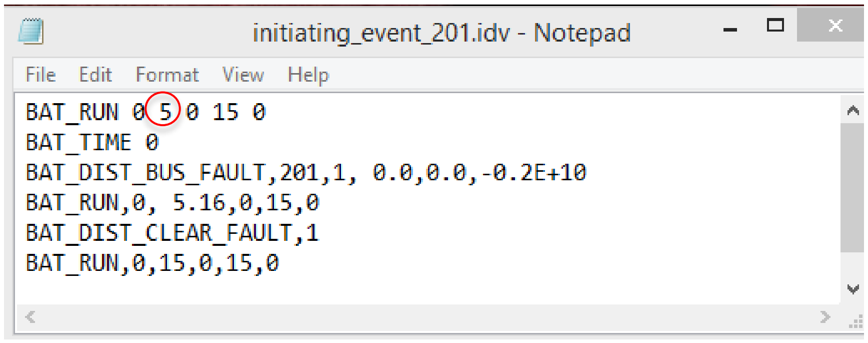
3.5 Simulate Contingencies

3.5.1 Traditional Dynamic Simulations (no cascading outages)

“RunFlatStart_Config.ini” and “RunTestFault.py” are the main modules involved in this step, and “RunFlatStart_Config.ini” provides the necessary settings for running “RunTestFault.py”, which can be called to simply conduct a dynamic simulation for a specified contingency. The parameters necessary for running “RunTestFault.py” in “RunFlatStart_Config.ini” are:

1. “savefile_testfault”: this is the “.sav” case file for the simulation, and it should be the file generated in Section 3.3.1.
2. “snapfile_testfault”: this is the “.snp” file for the simulation, and it should be the file generated in Section 3.3.2.
3. “log_file_testfault”: this specifies the name of the “.log” file to be generated.
4. “outfile_testfault”: this specifies the name of the “.out” file to be generated.
5. “Time_parameter_testfault”: this determines the time duration of the dynamic simulation.
6. “timestep_testfault”: this specifies the simulation time step for the dynamic simulation.
7. “filter_testfault”: this specifies the filter time constant used in calculating bus frequency deviations for the dynamic simulation.

8. “idvfile_testfault”: this parameter specifies the “.idv” file in which the contingency is applied. One typical “.idv” file is shown in Figure 3.3. In this contingency, the dynamic simulation first runs to 5 seconds, as circled in red, and this number can be modified if needed. Then the fault condition is described from line 3. User can find those details in PSS/E API [2]. The upper case letters are Batch command syntax.


initiating_event_201.idv - Notepad
File Edit Format View Help

```
BAT_RUN 0 5 0 15 0
BAT_TIME 0
BAT_DIST_BUS_FAULT,201,1, 0.0,0.0,-0.2E+10
BAT_RUN,0, 5.16,0,15,0
BAT_DIST_CLEAR_FAULT,1
BAT_RUN,0,15,0,15,0
```

Figure 3.3. Example of One Typical “.idv” File

After the parameter configurations are finished, “RunTestFault.py” should be called to run the dynamic simulation. The “.log” file, the “.out” files, and the “.csv” file, which will be generated in the main folder, can be checked to see the simulation results.

3.5.2 Modeling of Cascading Outages through DCAT Run

In the main code folder “Main_Data”, “set_DCAT_path.ini” and “MainDCAT.py” are involved in this step to run DCAT simulation. “set_DCAT_path.ini” provides the necessary path linking to contingency definition settings and output folders for running “MainDCAT.py”.

The corresponding program settings are located in the “Config.ini” file, in the example subfolders, e.g., “Examples\Test_system\Example1\”. After modifying the parameter configuration in “Config.ini”, if necessary, the Python module “MainDCAT.py” can be called to conduct a full run of the DCAT procedure, which conducts not only the dynamic simulations, but also potential cascading failure analysis for extreme contingencies. Section 3.5.2.2 talks about some further modifications that might be needed by the users; however, the modifications are not needed for a basic DCAT run.

3.5.2.1 Parameter Configurations in “Config.ini”

Before running “MainDCAT.py”, the following parameters could be modified in the file “Config.ini”:

Parameters related to dynamic simulation

- “Time_parameter”: this determines the time duration of the dynamic simulation.
- “MaxNumIter”: is the maximum iteration of dynamic simulation convergence that will be checked to verify if steady state is reached.
- “timestep1”: is the simulation time step.
- “filter1”: is the filter time constant used in calculating bus frequency deviations.

Parameters related to judging the steady state

- “TOL”: this determines the tolerance of judging whether the dynamic simulation has reached the steady state.
- “ck_settle_time”: for example, if ck_settle_time = 2, beyond the dynamic run for “Time_parameter” seconds, the simulation results of the previous 2 seconds will be scanned every 2 seconds, and the largest differences will be compared with “TOL” to judge whether or not the steady state has been reached.

Parameters related to judging the overloading

- “PER”: the threshold to judge overloading; e.g., if PER = 100, this means the threshold will be 100% of the line rating.
- “RATE_overloading”: defines which rating of the line will be adopted to judge the overloading: PCTMVARATEA stands for rateA, PCTMVARATEB stands for rateB, and PCTMVARATEC stands for rateC. Refer to [2] for other options. (Note: rate C is adopted here. During case preparation step, the Python module “Make_DYR_withPNNLrelays__xx.py” modifies the rate C of the branches to be the minimum of 130% rate A and 115% rate B, based on common industry practice for longer time allowed overloading).

Parameters related to the PSS®E base case files

- “savefile”: this is the “.sav” case file for the simulation, and it should be the file generated in Section 3.3.1. Note that the path, as well as the file name, should be provided, if the file is not located in the same folder as “MainDCAT.py”.
- “snapfile”: this is the “.snp” file for the simulation, and it should be the file generated in Section 3.3.2. Note that the path, as well as the file name, should be provided, if the file is not located in the same folder as “MainDCAT.py”.

Parameter related to the file needed for running the inertia power flow

- “inlfile”: this specifies the “.inl” file that is needed for running the PSS®E inertial power flow (INLF). Make sure that the correct “.inl” file exists in the same folder as “MainDCAT.py” does.

Parameter related to the file needed for running the inertia power flow

- “sub_file”: this specifies the name of the “.sub” file that is to be adopted for the ACCC process. This file will also be used to generate a subsystem, where the overloading check is conducted. Make sure that the correct “.sub” file exists in the same folder as “MainDCAT.py” does.
- “mon_file”: this specifies the name of the “.mon” file that is going to be adopted for ACCC process.

Parameter related to the initial contingency file

- “idvfile”: this specifies the “.idv” file that describes the initial contingency (initiating event) in the first iteration of dynamic simulation.

Parameter specifying the name of the “.out” file to be generated

- “outfile”: this specifies the name of the “.out” file.

Parameter specifying the names of the ".log" files to be generated

The following parameters specify the names of the ".log" files to be generated. If the names are set to be the same, this means the process of all iterations of a DCAT run will be recorded within one log file. They can also be set differently to record the different processes.

- "logfile": (logfile: initiating_event.log)

  This ".log" file records the information of the first iteration of dynamic simulation and also the SPS checking process.

- "logfile2": (logfile2: initiating_event_promptoutput.log)

  this ".log" file records the prompt information of the first iteration of dynamic simulation and the associated SPS checking process.

- "logfile_corr": (logfile_corr: initiating_event.log)

  this ".log" file records the ACCC process during the first iteration of a DCAT run.

- "logfile_after": (logfile_after: initiating_event.log)

  this ".log" file records the process of the later iterations of a DCAT run.

- "logfile2_after": (logfile2_after: initiating_event_promptoutput.log)

  this ".log" file records the prompt information in the later iterations of a DCAT run.

Parameter specifying the names of the temporary files or intermediate files

The following parameters set the names of some temporary files or intermediate files produced during the DCAT run. They do not need to be modified if DCAT is not to be run for particular purposes.

- Parameters related to temporary files in the SPS checking process

  – "SPS_logfile",:(SPS_logfile: test_SPS.log)

  This log file records the SPS checking process in the first iteration.

  – "SPS_logfile_after": (SPS_logfile_after: test_SPS.log)

  This log file records the SPS checking process for the later iterations.

  Both these SPS log files can be specified with the same name or with different names to record separately. These log files will be finally copied and written into the total log file.

  – "spsfile": (spsfile: SPS_action1.idv)

    This file specifies the .idv file to record the SPS action in the first iterations of DCAT runs

  – "spsfile_after": (spsfile_after: SPS_action1_aftercontingency.idv)

    This file specifies the .idv file to record the SPS action in the later iterations of DCAT runs

  – "idvfile_react_SPS",:(idvfile_react_SPS: idvfile_react_SPS.idv)

    This file specifies the ".idv" file generated, in the first iterations of DCAT runs, to reflect the SPS action to be utilized in the subsequent dynamic simulation.

  – "idvfile_react_SPS_after": (idvfile_react_SPS_after: idvfile_react_SPS_aftercontingency.idv)

This file specifies the ".idv" file generated, in the later iterations of DCAT runs, to reflect the SPS action to be utilized in the subsequent dynamic simulation.

– "returnfile_sps": (returnfile_sps: \Cases\AfterDynamicSim_sps.sav)

  This file specifies the ".sav" file returned after the SPS checking, in the first iterations of DCAT runs, is finished.

– "returnfile_sps_after" (returnfile_sps_after: \Cases\AfterDynamicSim_sps_aftercontingency.sav):

  This file specifies the ".sav" file returned after the SPS checking, in the later iterations of DCAT runs, is finished. This file will be used in the subsequent dynamic simulation immediately after the SPS checking.

- Parameters related to temporary files in the ACCC process
  – "confile": (confile: temp.con) To specify the name of the ".con" files to be generated for ACCC process.
  – "dfxfile": (dfxfile: test_corrective_actions.dfx)

    This file specifies the ".dfx" file in order to run ACCC analysis in the first iteration of DCAT run.
  – "dfxfile_after": (dfxfile_after:test_corrective_actions.dfx)

    This file specifies the ".dfx" file in order to run ACCC analysis in the later iterations of DCAT run. These files can be either set to the same or different names according to the purposes.
- Parameters related to intermediate case files (These case files are saved in the process of the DCAT runs).
  – "returnfile": (returnfile: \CASEs\After_dynamic_raw_file.raw)

    The file specifies the name of the ".raw" file returned after the dynamic simulation in the first iteration of DCAT.
  – "returnfile_sav" (returnfile_sav: \CASEs\AfterDynamicSim_corr.sav)
  – or "returnfile_raw": (returnfile_raw: \CASEs\AfterDynamicSim_corr.raw)

    these specify the network files to be returned after the ACCC in the first iteration of DCAT. The "returnfile_raw" file is going to be utilized in the subsequent iterations of DCAT.
  – "savefile_after": (savefile_after: \CASEs\After_dynamic_raw_file2_con.sav)

    This ".sav" file specifies the converted version of the of the "returnfile_raw" or "returnfile_raw_after".
  – "idvfile_reoverflow": (idvfile_reoverflow: idvfile_reoverflow.idv)

    specifies the ".idv" file generated to switch off the overloading branch during the dynamic simulation.
  – "returnfile_after": (returnfile_after: \CASEs\SAVNW_After_dynamic_raw_file_aftercontingency.raw)

    This file specifies the file generated after the dynamic simulation in the later iterations of a DCAT run.
  – "casefile_after": (casefile_after: \CASEs\AfterDynamicSim_sps_aftercontingency1.sav)

This “.sav” file specifies the file generated after checking the SPS in the later iterations of a DCAT run.

– “returnfile_sav_after”: (returnfile_sav_after: \CASEs\AfterDynamicSim_corr_aftercontingency.sav)

– “returnfile_raw_after”: (returnfile_raw_after: \CASEs\AfterDynamicSim_corr_aftercontingency.raw)

these files specify the “.sav” and “.raw” files returned after ACCC analysis in the later iterations of a DCAT run.

After the configurations of the parameters in the “Config.ini” are complete, the “MainDCAT.py” can be executed to simulate the potential cascading failure analysis in extreme contingencies.

3.5.2.2 Further Modifications

This section describes some further modifications that might be needed by users for other purposes; the modifications are not necessary for a basic DCAT run.

1. Modify the SPS look-up table

Modification of the SPS look-up table can be done within the subroutine “Make_SPS”.

- The parameters “from_bus”, “to_bus”, “to_bus1”, and “”ckt_id” specify the elements whose flow will be checked.
- The parameter “str” specifies whether to check “AMPS” or “MVA” on that element.
- “max_value” and “min_value” specify the limitations on each element.
- “SPS_type”, “SPS_action”, “SPS_from_bus”, “SPS_to_bus”, “SPS_mach_bus”, and “SPS_ckt_id” specify the SPS action that is going to be adopted accordingly for each element.

2. Modify the ACCC parameters

The parameters related to conducting the ACCC can be modified through the parameters in application program interface (API) “psspy.accor_3” in the subroutine “Test_CorrectiveAction”. Note that the API “psspy.accor_3” appears twice in that subroutine and the parameters should be modified in both occurrences of that API.

3.6 DCAT Examples and Simulation Results on a PSS/E Test System

This section presents simulation results for DCAT on several examples using a test cases (savnw.sav) that is provided with the PSS/E software package. The purpose of these examples is to show the importance of performing hybrid dynamic and steady-state simulations with protection modeling to accurately mimic the cascading outage process. They also show how planning engineers can use DCAT for cascading-outage analysis and how the results are reported. This test system has 23 buses and six power plants. Table 3.2compares different examples. Five different simulation tests are performed using the test system, and outcomes of each test are presented in the following subsections. Figure 3.4 shows a one-line diagram of the test system.

Table 3.2. Comparison between Simulation Examples

Example Section No.	System Type	Initiating Event	First Dynamic Simulation Reaches a Stable Point	SPS/RAS Activated	Corrective Action Needed	Second Dynamic Simulation Reaches a Stable Point	Generator and Load Outage
5.1.1	PSS/E test system “savnw”	3 phase line fault	Yes (30 s) 2 relay actions	N/A	No	N/A	None
5.1.2	PSS/E test system “savnw”	3 phase line fault with distance relay failed to send transfer trip signal	Yes (30 s) 2 relay actions	N/A	No	N/A	None
5.1.3	PSS/E test system “savnw”	3 phase bus fault for10 cycles	Yes (75 s) 2 relay actions	N/A	No	N/A	Gen loss = 600 MW Load loss = 0 MW
5.1.4	PSS/E test system “savnw”	3 phase bus fault for 12 cycles	No (blackout)	N/A	N/A	N/A	Gen loss = 3,259 MW (before system collapses)
5.15	PSS/E test system “savnw”	3 phase bus fault for 6 cycles	Yes (25 s) 5 relay actions	Yes	No	Yes (25 s) No relay actions	None

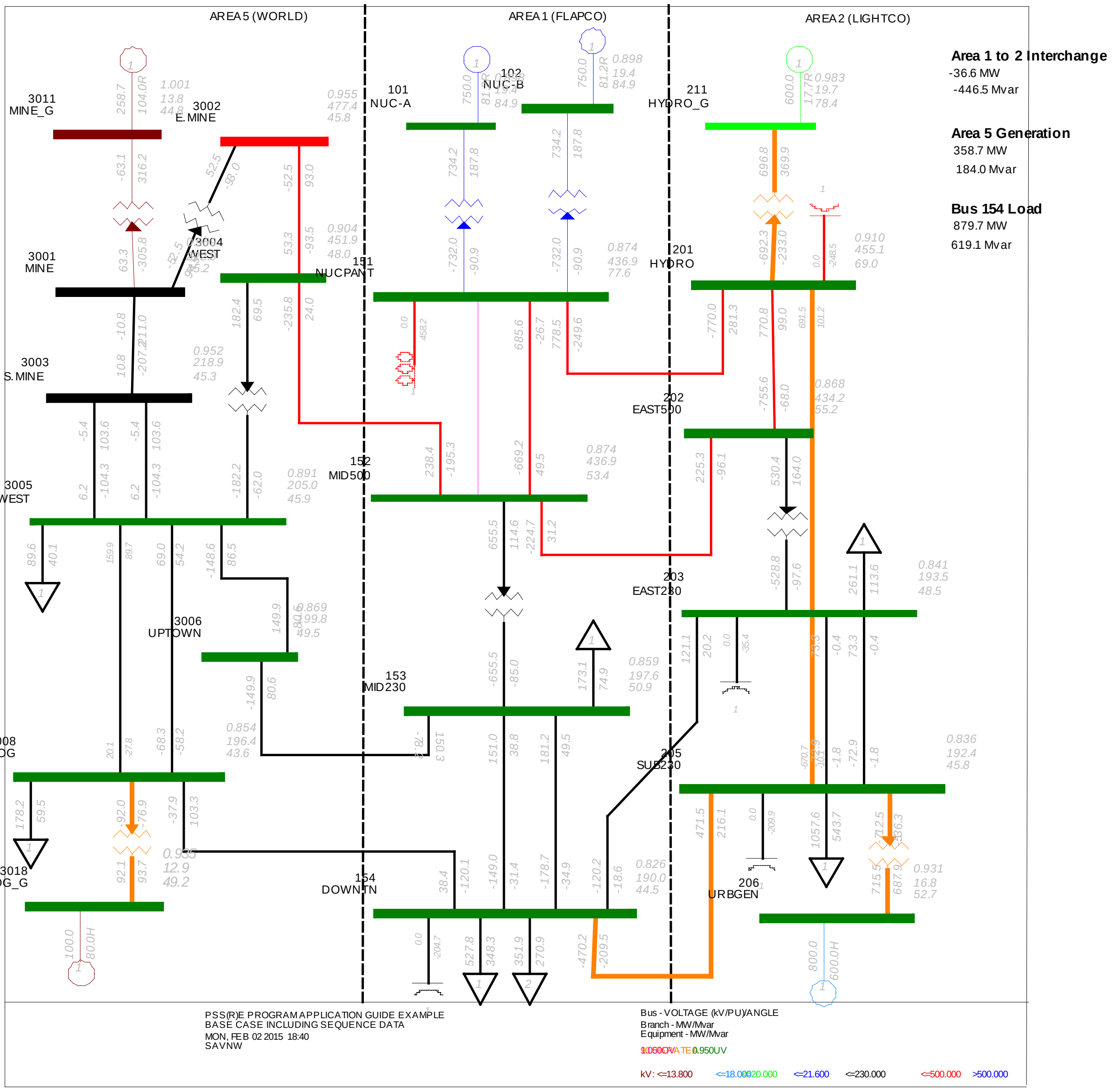


Figure 3.4. One-Line Diagram of the Test System

3.6.1 Example 1: Not a Close-In Fault in Pilot Scheme Line – Using Fictitious Node

A line fault is applied on one of the lines connected to Bus 152 at a distance of 90% from it. Distance relays are modeled on both ends of the line with an ability to send a transfer trip to the other end upon sensing a Zone 1 fault. Though the other end of Line 152 sees a Zone 2 fault, this pilot scheme trips the breaker as soon as the other relay on Bus 152 times out on the Zone 1 fault. Upon successful operation of both breakers, the fault is isolated, and there are no other tripping actions.

To model a fault in PSS/E at any location in a transmission line other than the two line ends, a fictitious node needs to be added, as explained in Appendix B. In this test, to model a fault in the line connecting Buses 151 and 152 that is located at a distance of 0.1 pu of total line length from Bus 151, a new fictitious node (151152) is added between Buses 151 and 152. Figure 3.5 shows the location of the fictitious bus.

Distance relays then need to be associated with the two branches newly created by the fictitious bus addition. That is, one branch is from the near end to the fictitious bus, and the other is from the remote end to the fictitious bus.

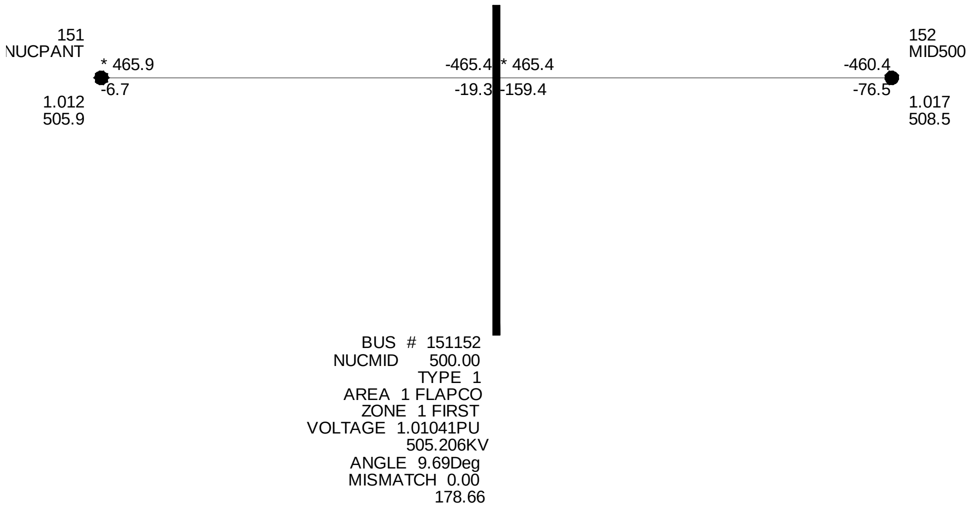


Figure 3.5. A Fictitious Bus between Buses 151 and 152

The bus fault is introduced at the fictitious bus (151152) at *t* = 5 s and simulation runs until dynamic simulation reaches a steady state. In this test, dynamic simulation reaches a steady state at *t* = 16 s. The following is the sequence of relay tripping events:

1. Distance relay (DISTR1) at Circuit 1 from 151 to 151152 is activated as Zone 1 and its timer started at *t* = 5 s.
2. Distance relay (DISTR1) at Circuit 1 from 152 to 151152 is activated as Zone 2 and its timer started at *t* = 5 s.
3. Zone 1 timer timed out at *t* = 5.017 s; self-trip breaker timer and also transfer trip and breaker trip timers started at the same time.
4. Circuit 1 from 151 to 151152 tripped at *t* = 5.05 s and transfer trip timer also timed out at the same time. In this case, the Zone 1 relay accelerates the other relay, and as a result, the other end (Circuit 1 from 152 to 151152) trips at the same time (*t* = 5.05 s), and soon thereafter the two voltages start to recover.

The channel plot in Figure 3.6 shows that the voltage at Bus 151 collapses more than the Bus 152 voltage. This indicates that the fault is closer to Bus 151.

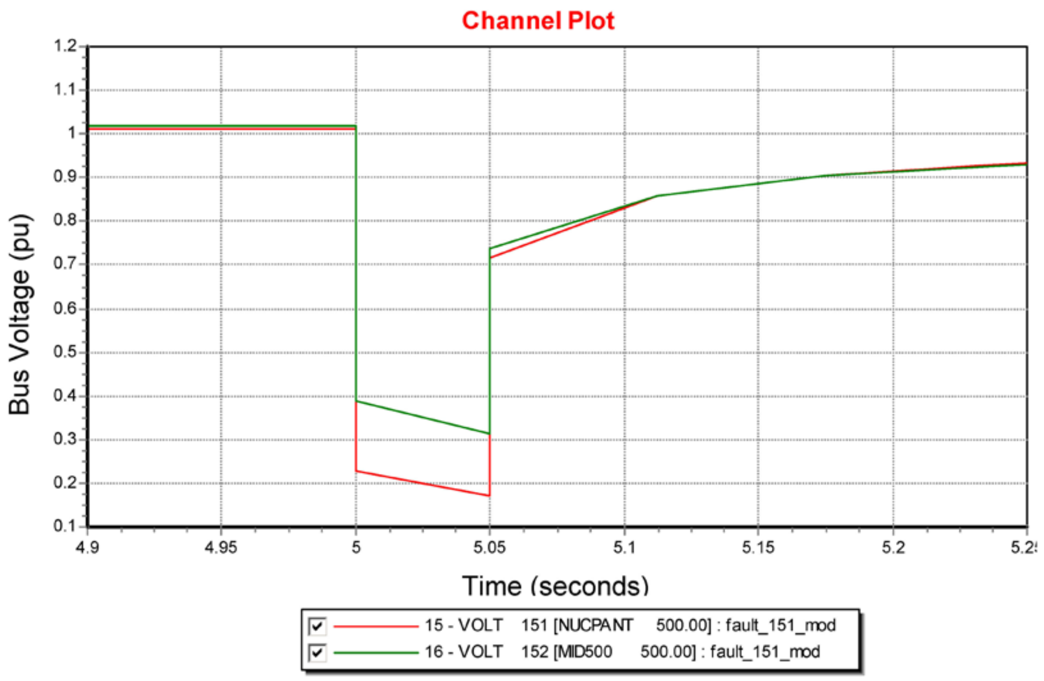


Figure 3.6. Voltage Plots of the Terminal Buses of the Faulted Line for Test 1

3.6.2 Example 2: Not a Close-In Fault in Step Distance Line – Using Fictitious Node

This simulation uses the same procedure and files that were used in Test 1 except that the transfer trip capability of DISTR1 is assumed to have failed. As a result of that, the near end of the line to the fault at Bus 151 trips on the Zone 1 setting (4 cycles) and the other end of the line at Bus 152 trips at the Zone 2 setting (22 cycles).

Each end will trip according to the Zone 1 or Zone 2 delays where appropriate. The bus fault is introduced at the fictitious bus (151152) at $t = 5$ s and simulation runs until dynamic simulation reaches a steady state. In this test, dynamic simulation reaches a steady state at $t = 16$ s. The following is the sequence of relay tripping events:

1. Distance relay (DISTR1) at Circuit 1 from 151 to 151152 is activated as Zone 1 and its timer started at $t = 5$ s.
2. Distance relay (DISTR1) at Circuit 1 from 152 to 151152 is activated as Zone 2 and its timer started at $t = 5$ s.
3. Zone 1 timer timed out at $t = 5.017$ s; self-trip breaker timer and breaker timer started at the same time.
4. Circuit 1 from 151 to 151152 tripped at $t = 5.05$ s.
5. Circuit 1 from 152 to 151152 trips as Zone 2 fault at $t = 5.333$ s and the channel plot (Figure 3.7) shows the two voltages start to recover after tripping both ends of the branch.

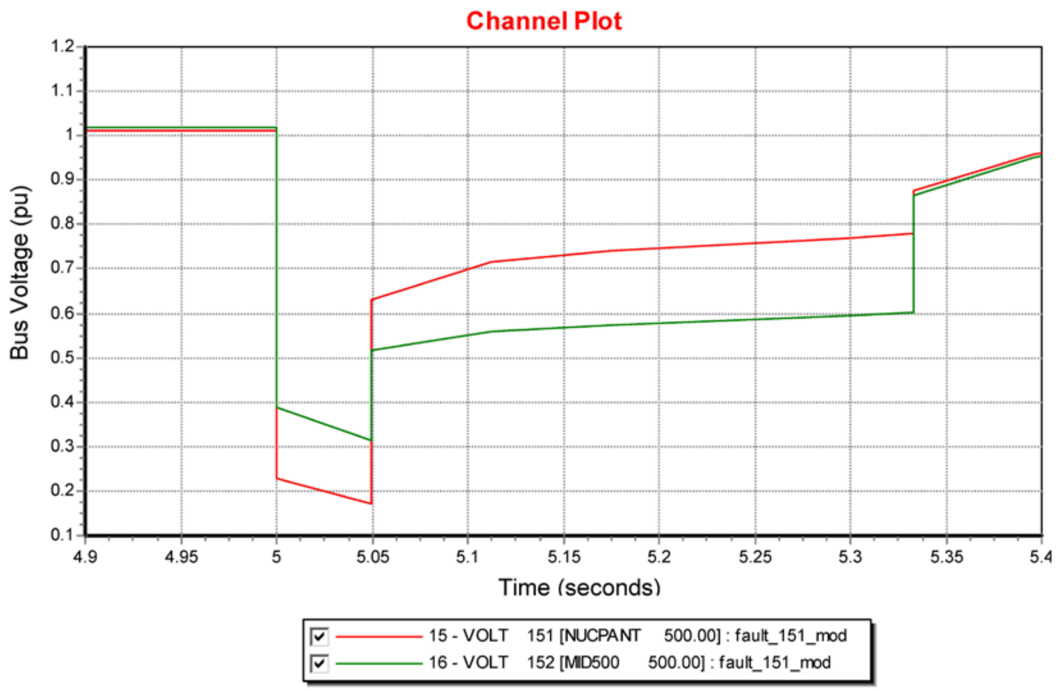


Figure 3.7. Channel Plot for Test 2

3.6.3 Example 3: Bus Fault

In Test 3, a fault is applied at Bus 201 at t = 5 s and the fault is cleared after 10 cycles. The simulation runs until dynamic simulation reaches a steady state. In this test, dynamic simulation reaches a steady state at t = 75 s. Table 3.3 shows a summary. No corrective action was required for this contingency with these protection settings. The details of each tripping action in Test 3 are presented in Table 3.4. Simulation result plots are shown in Figure 3.8 and Figure 3.9.

Table 3.3. Relay Trips Summary of Test 3

Relay Type										
DISTR1	TimeOut	Busfrom	Busto	ckt						
DISTR1	5.1	201	202	1						
VTGTPA	TimeOut	Bus	BusName	BuskV	Pgen (MW)	Qgen (MVAr)	GenBus	GenID	GenName	GenkV
VTGTPA	5.237	211	HYDRO_G	20	600	17.75	211	1	HYDRO_ G	20

ckt = circuit
Pgen = generator real power
Qgen = generator reactive power

Table 3.4. Tripping Action Details of Test 3

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	Details
DISTR1	5.1	201	202	1	• Distance relay (DISTR1) at circuit 1 from 201 to 202 is activated as Zone 1 and its timer started at *t* = 5 s. • Zone 1 timer timed out at *t* = 5.067 s; self-trip breaker timer started at the same time. • Circuit 1 from 201 to 202 is tripped at *t* = 5.1 s. • Channel plots for Bus 201 and 202 are shown in Figure 3.8.
VTGTPA	TimeOut	Bus	BusName	BuskV	Details
VTGTPA	5.237	211	MINE_G	20	VTGTPA at Bus 211: • Pickup timer started at *t* = 5.004 s. • Breaker timer started at *t* = 5.154 s. • Breaker timer timed out at time *t* = 5.237 s. • Channel plot for Bus 211 is shown in Figure 3.9 • Voltage at Bus 211 starts to recover after tripping and reached a steady state around 60 s.

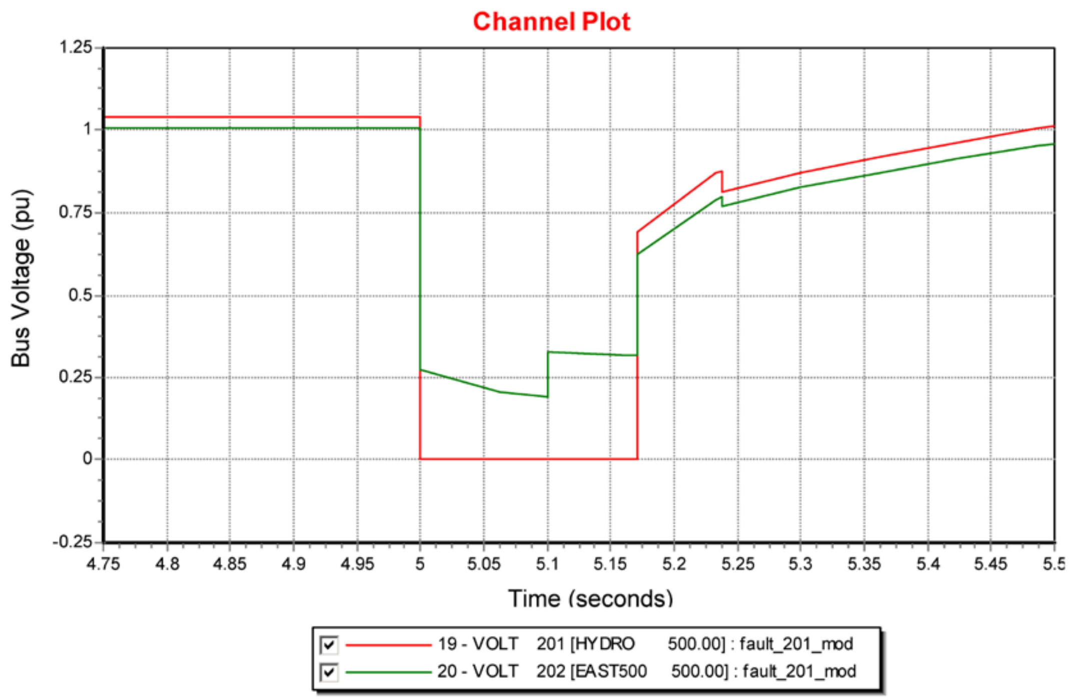


Figure 3.8. Channel Plot for Voltages at Buses 201 and 202

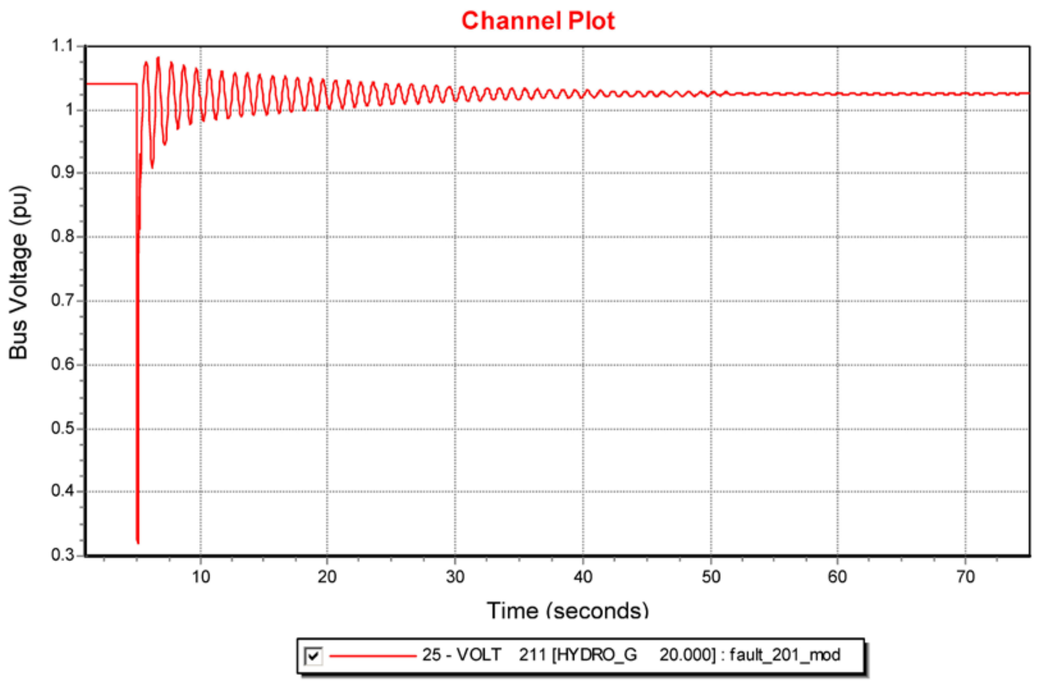


Figure 3.9. Channel Plot for Voltage at Bus 211

3.6.4 Example 4: Bus Fault Leads to Blackout

In this dynamic simulation, a fault is applied at Bus 151 at *t* = 5 s; the fault is applied for 12 cycles and then cleared. A significant number of undervoltage and underfrequency generator relays were tripped due to this fault, which leads to system blackout. The network did not converge after *t* = 6.3708 s. A total of seven relays are activated during this dynamic simulation; Table 3.5 shows a summary.

Table 3.5. Relay Trips Summary of Test 4

Relay Type										
DISTR1	TimeOut	Busfrom	Busto	ckt						
DISTR1	5.1	151	152	1						
DISTR1	5.1	151	152	2						
VTGTPA	TimeOut	Bus	BusName	BuskV	Pgen (MW)	Qgen (MVAr)	GenBus	GenID	GenName	GenkV
VTGTPA	5.237	101	NUC-A	21.6	750	81.19	101	1	NUC-A	21.6
VTGTPA	5.237	102	NUC-B	21.6	750	81.19	102	1	NUC-B	21.6
FRQTPA	TimeOut	Bus	BusName	BuskV	Pgen	Qgen	GenBus	GenID	GenName	GenkV
FRQTPA	6.362	3018	CATDO G_G	13.8	100	80	3018	1	CATDOG_G	13.8
FRQTPA	6.371	206	URBGEN	18	800	600	206	1	URBGEN	18
FRQTPA	6.371	3011	MINE_G	13.8	258.66	104.04	3011	1	MINE_G	13.8
FRQTPA	6.383	211	HYDRO_ G	20	600	17.75	211	1	HYDRO_G	20

The sequence of tripping is shown in Figure 3.10. It is observed that Tripping 1 is due to a distance relay, Tripping 2 is due to undervoltage at Generators 101 and 102, and the remaining trippings from 3 to 5 are due to underfrequency at Generators 3018, 206, 3011, and 211. The details of each tripping action in Test 4 are presented in Table 3.6. Simulation result plots are shown in Figure 3.11 and Figure 3.12.

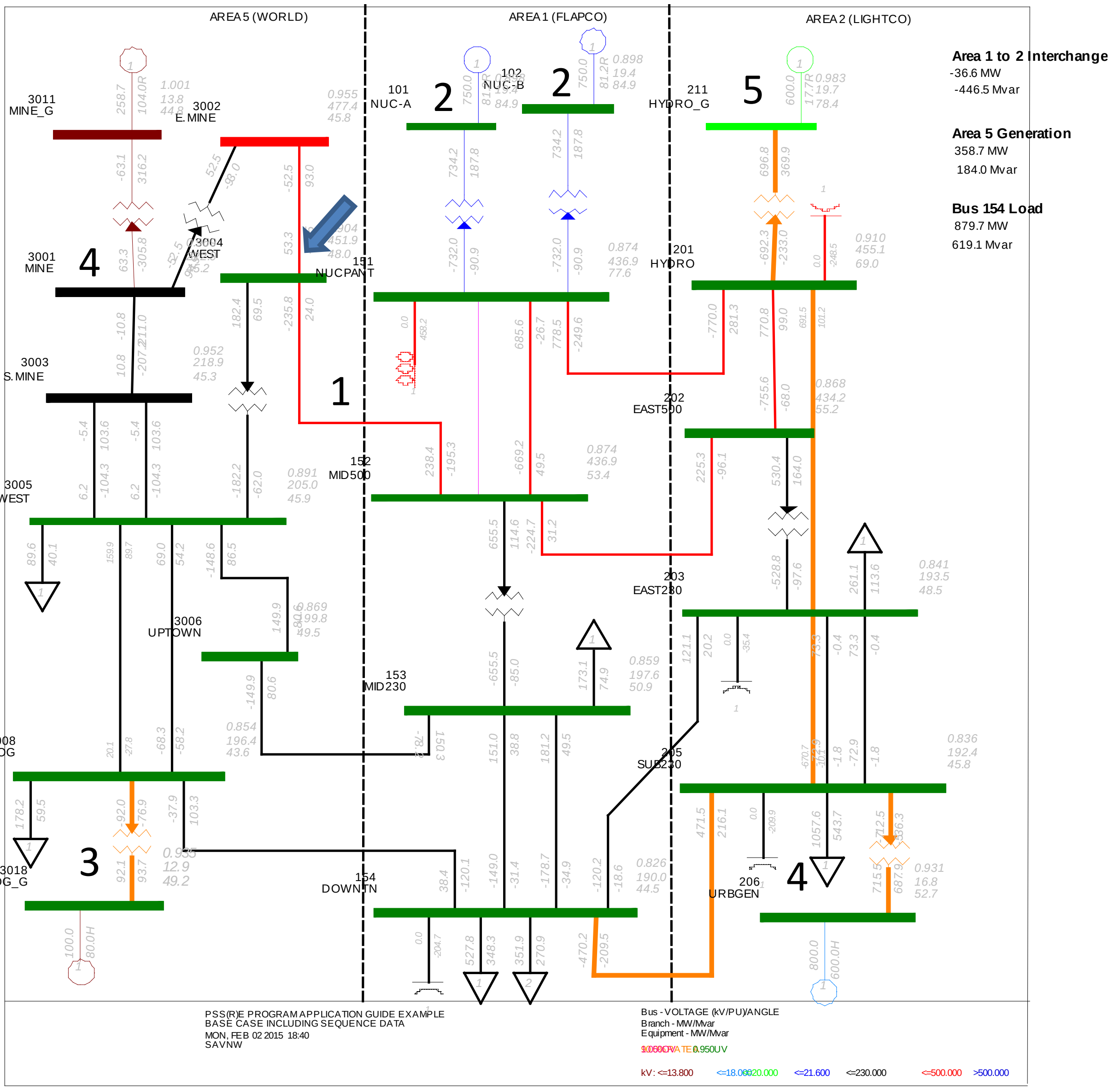


Figure 3.10. One-Line Diagram of Test System to Show Sequence of Tripping

Table 3.6. Tripping Action Details of Test 4

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	Details
DISTR1	5.1	151	152	1	• Distance relay (DISTR1) at Circuit 1 from 151 to 152 and relay at Circuit 2 from 151 to 152 are activated as Zone 1 and their timers started at *t* = 5 s. • Zone 1 timer timed out at *t* = 5.067 s; self-trip breaker timer started at the same time. • Circuit 1 from 151 to 152 and Circuit 2 from 151 to 152 are tripped at *t* = 5.1 s.
DISTR1	5.1	151	152	2	
VTGTPA	TimeOut	Bus	BusName	BuskV	Details
VTGTPA	5.237	101	NUC-A	21.6	VTGTPA at Buses 101 and 102: • Pickup timer started at *t* = 5.004 s. • Breaker timer started at *t* = 5.154 s. • Breaker timer timed out at time *t* = 5.237 s. • Channel plots for Buses 101 and 102 are shown in Figure 3.11.
VTGTPA	5.237	102	NUC-B	21.6	
FRQTPA	TimeOut	Bus	BusName	BuskV	Details
FRQTPA	6.362	3018	CATDOG_G	13.8	FRQTPA at Bus 3018: • Pickup timer started at *t* = 6.275 s. • Breaker timer started at *t* = 6.279 s. • Breaker timer timed out at time *t* = 6.362 s.
FRQTPA	6.371	206	URBGEN	18	FRQTPA at Buses 206 and 3011: • Pickup timer started at *t* = 6.283 s. • Breaker timer started at *t* = 6.287 s. • Breaker timer timed out at time *t* = 6.362 s.
FRQTPA	6.371	3011	MINE_G	13.8	
FRQTPA	6.383	211	HYDRO_G	20	FRQTPA at Bus 3011: • Pickup timer started at *t* = 6.296 s. • Breaker timer started at *t* = 6.300 s. • Breaker timer timed out at time *t* = 6.383s. • Channel plots for speeds of machines 3018, 206, 3011, and 211 are shown in Figure 3.12.

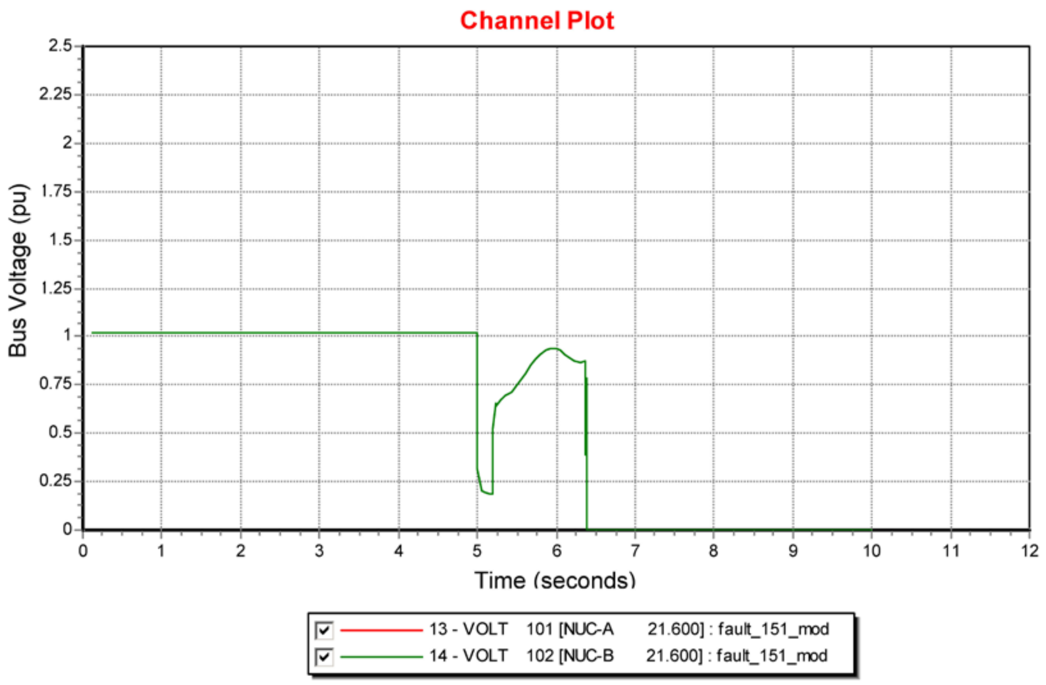


Figure 3.11. Channel Plots for Voltages at Buses 101 and 102

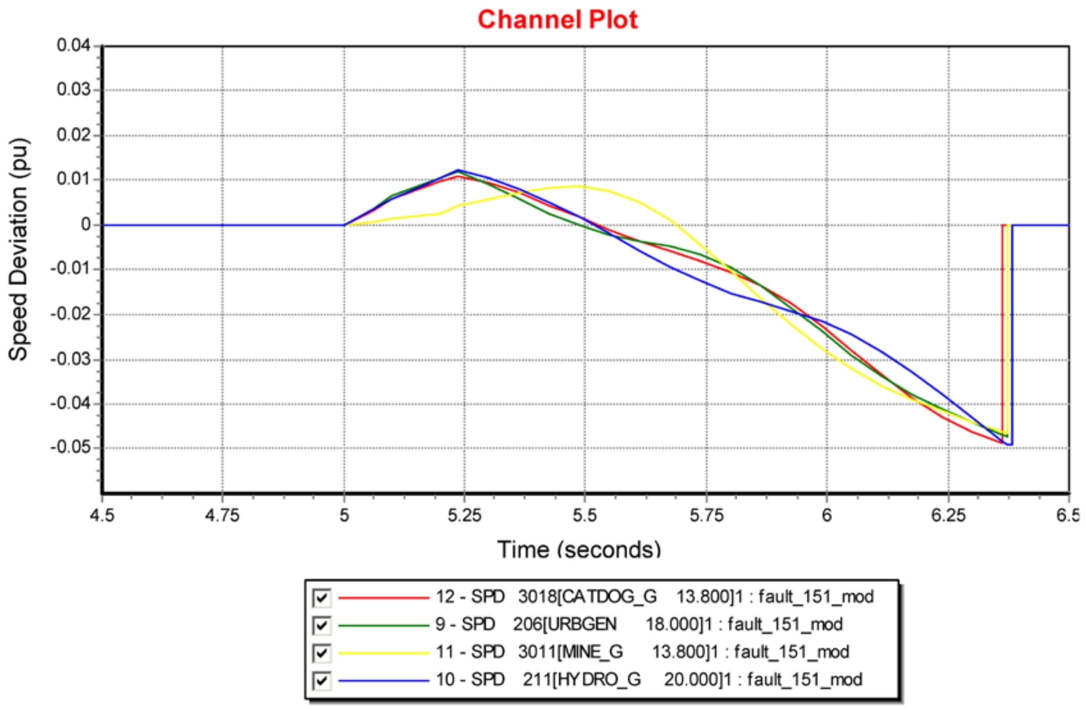


Figure 3.12. Channel Plots for Speeds of Machines 3018, 206, 3011, and 211

3.6.5 Example 5: Activation of an SPS/RAS

In this example, a bus fault that lasts for six cycles was introduced at Bus 203, which was then tripped to isolate the fault.. This was one of the extreme events that had the potential to trigger an SPS/RAS. The fault was introduced at *t* = 5 seconds and the bus was isolated after 6 cycles, along with a line trip during the dynamic simulation. There were no relay tripping observed during the dynamic simulation. A graph of the simulation result is shown in Figure 3.13.

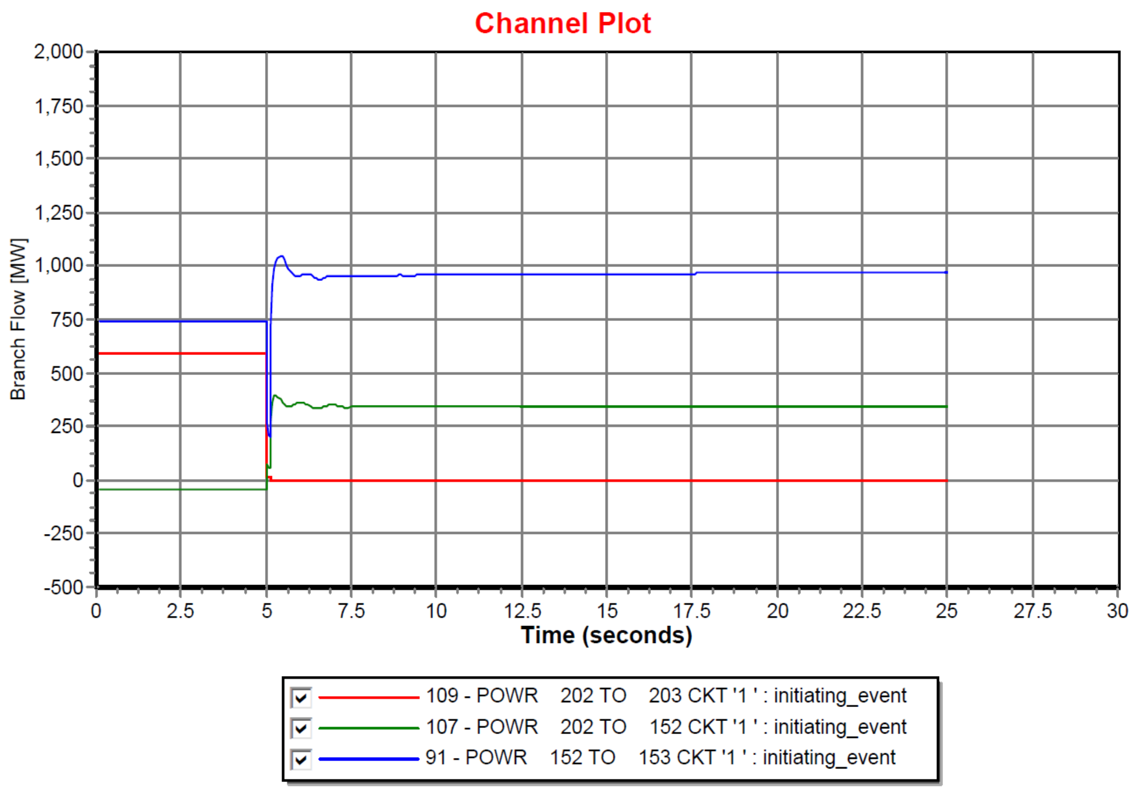


Figure 3.13. Example 5: Branch Flows

After the dynamic simulation, one control condition that could trigger an SPS/RAS was observed in the post-dynamic steady-state case. A second dynamic simulation has been performed to trigger this cascading event.

No other trippings have been observed during the dynamic simulation where the SPS/RAS event has been triggered. The line overloads observed on the system were below 130% of Rate A and no voltage violations below 0.9 pu were observed. No corrective action was required for this contingency with these protection settings. The sequence of DCAT actions that were performed for this contingency is shown in Figure 3.14. This contingency resulted in no tripping actions, generation loss load loss, as given in Table 3.7.

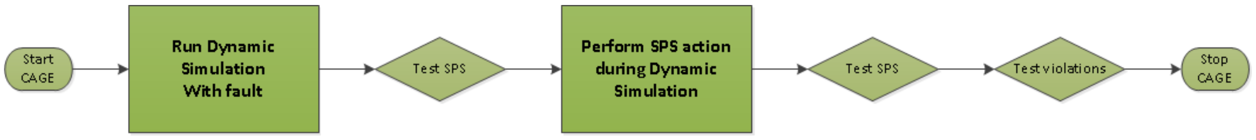

Start CAGE
Run Dynamic Simulation With fault
Test SPS
Perform SPS action during Dynamic Simulation
Test SPS
Test violations
Stop CAGE

Figure 3.14. Sequence of Events Performed by DCAT for Example 4

Table 3.7. Generation and Load Loss Summary for Example 5

Load loss (MW)	0
No. of Total Tripping Actions	0
No. of SPSs/RASs Triggered	1
No. of Overloaded Lines	0
Corrective Actions	None

4.0 Parallel Computing of DCAT Run

In reality, we may need to simulate the potential consequences of a large number of contingencies, and therefore, the total running time might become extremely long. To shorten the total simulation time, the parallel computing technique is adopted here. The main idea of the parallel computing of a DCAT run with a large number of contingencies can be illustrated in Figure 4.1. The package called “MPjobs”, a tool developed by ERCOT [3] to run PSS®E scripts in parallel, is used to parallelize the computation of DCAT processes.

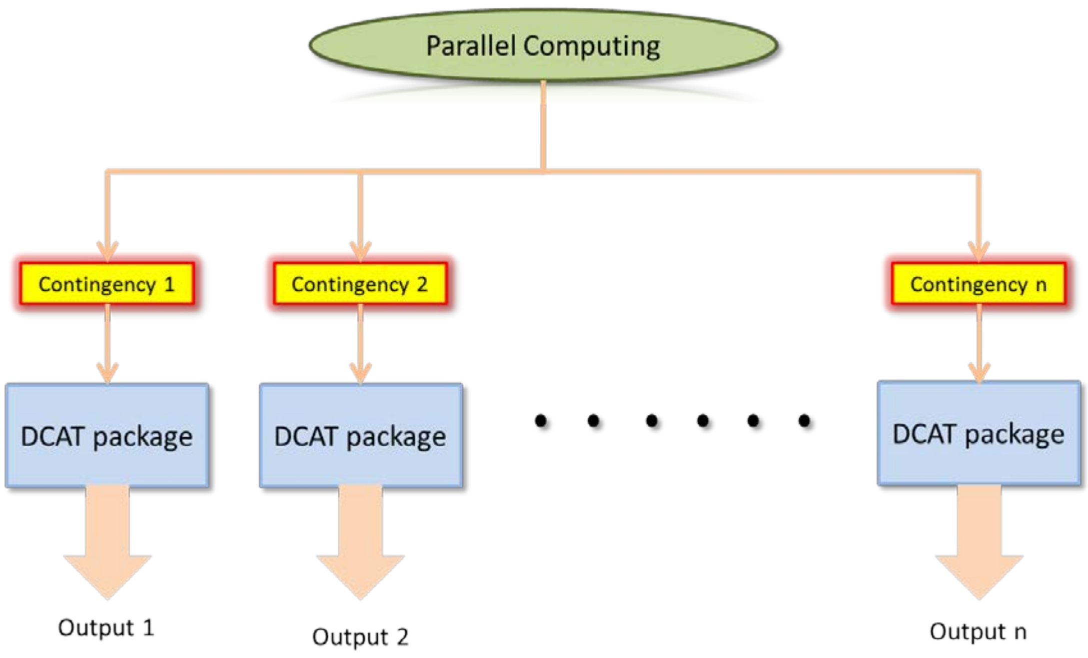


Figure 4.1. Main Idea of DCAT Parallel Computing

Figure 4.2 shows a preliminary test result of parallel computing that is based on the package “MPjobs”. Dynamic simulations are conducted in 1,000 contingency scenarios by using different processors; (when just one processor is used, the whole process is just like conventional computing without parallel computing). The maroon line in Figure 4.2 denotes the time needed to finish the whole process; the green line illustrates the scalabilities when different numbers of processors are adopted. Scalability (*n*) = (Time needed when one processor is adopted) ÷ (Time needed when *n* processors are adopted)

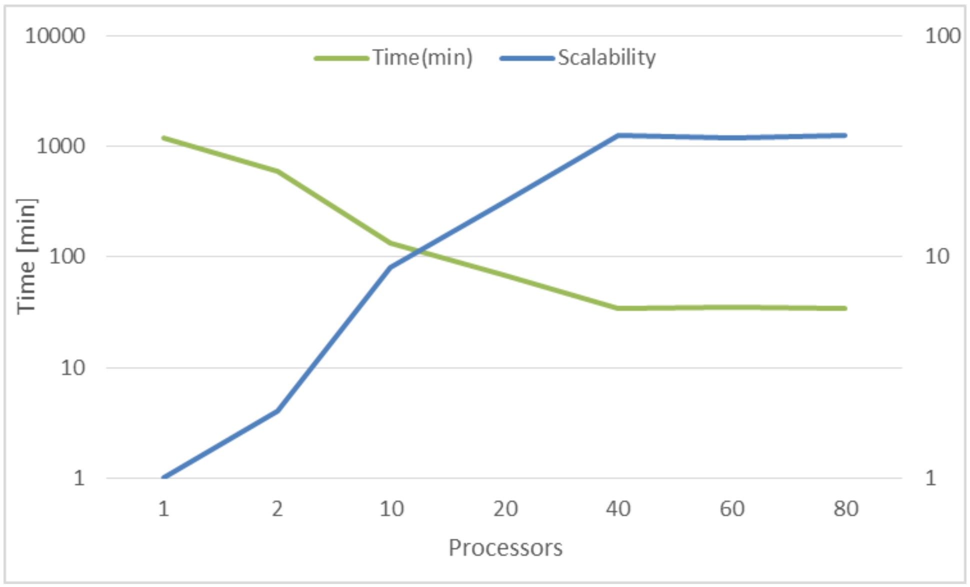


Figure 4.2. Preliminary Test Results of “MPjobs”

4.1 Basic Structure and Flow of “MPjobs”

The package “”MPjobs” is mainly composed of the following files:

- the “.ini” file: this file gives the information for the package to run;
- the main Python script to run PSS®E simulation;
- the main function “mpjobs.py”;
- the files and lists describing the X, Y and Z variables; The variable(s) that change with every run are assigned to them (X vars Y vars, and Z vars).
- other supporting files such as the files to be imported, the files needed for running ACCC, etc.

The basic flow is as follows: the main function “mpjobs” first reads in the “.ini” file to extract information including how many processors to use, how much delay to apply, the main Python script to call to run PSS®E simulation, the X, Y, and Z variables and the associated files, etc. Then it searches for an available processor, and assigns a scenario based on the combination of the X, Y, Z variables. After that, it calls the main Python script to run PSS®E simulations with the assigned variable files and scenarios, and stores the outputs in the designated places. This basic flow can be shown as in Figure 4.3.

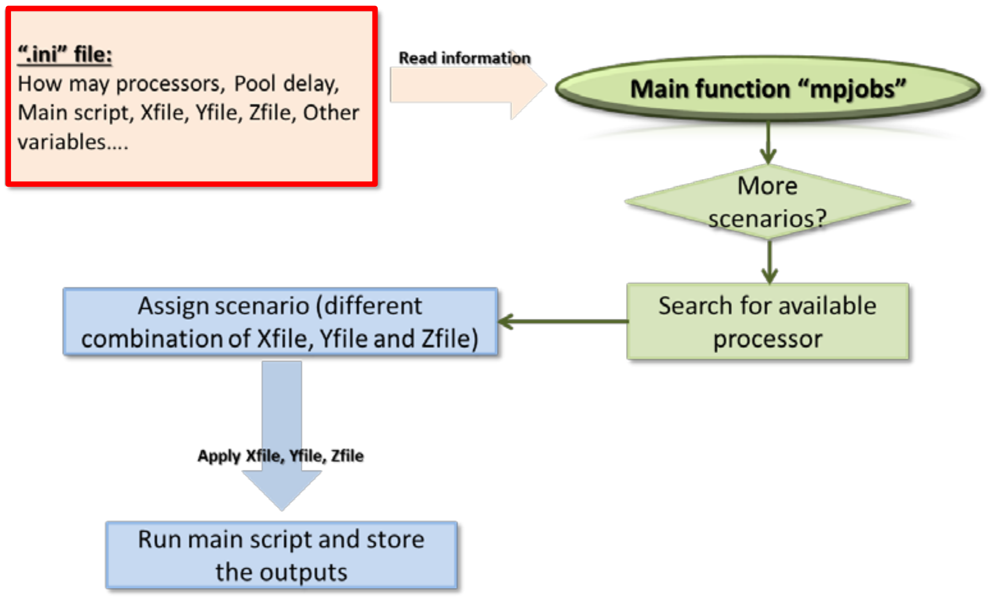

".ini" file:
How may processors, Pool delay, Main script, Xfile, Yfile, Zfile, Other variables....
Read information
Main function "mpjobs"
More scenarios?
Search for available processor
Assign scenario (different combination of Xfile, Yfile and Zfile)
Apply Xfile, Yfile, Zfile
Run main script and store the outputs

Figure 4.3. Basic Flow of "MPjobs"

4.2 Configurations Needed for "MPjobs" to Run DCAT

Usually, there is no need to modify the main function "mpjobs.py". The configurations are mainly related to the main Python script to run PSS®E simulation, the files and lists describing the variables, and the ".ini" file.

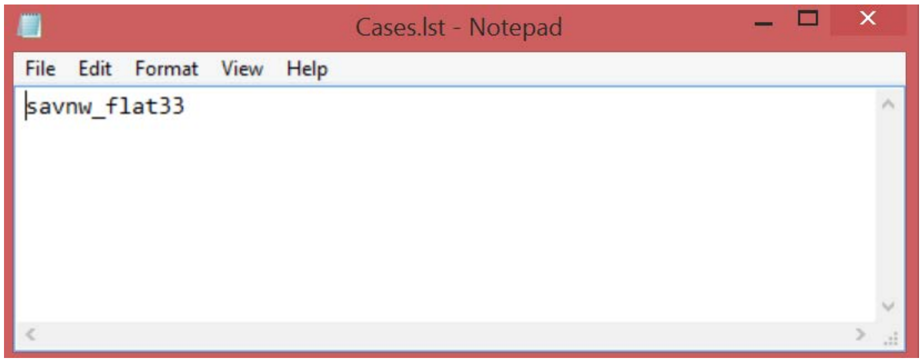
4.2.1 Scenario Setup

The "MPjobs" package can handle three loops of different variables (xvars, yvars, and zvars) at most, and the different combinations of these three variables form different scenarios. For example, if xvars has two cases, yvars has three cases, and zvars has four cases, the total number of scenarios is $2 \times 3 \times 4 = 24$.

The parallel computing of the DCAT process is to simulate the potential consequences of the same power system in facing different contingencies. We can either use one loop of variable (xvars = different contingencies) or two loops of variables (xvars = base cases, yvars = different contingencies). Here we choose to use two loops of variables, and in this situation, the simulation can be easily expanded so it can be conducted on different power systems by adding more system base cases to xvars.

4.2.1.1 xvars Setup

To set up the xvars, simply establish an ".lst" file and put the system name in it. The system files need to be put in the same folder as that ".lst" file. For example, we generate a file named "cases.lst", in which the system name "SAVNW_Finalv33" is written. Then the system files "SAVNW_Finalv33.snp" and "SAVNW _Finalv33_con.sav" are also put in the same folder as the file "cases.lst".



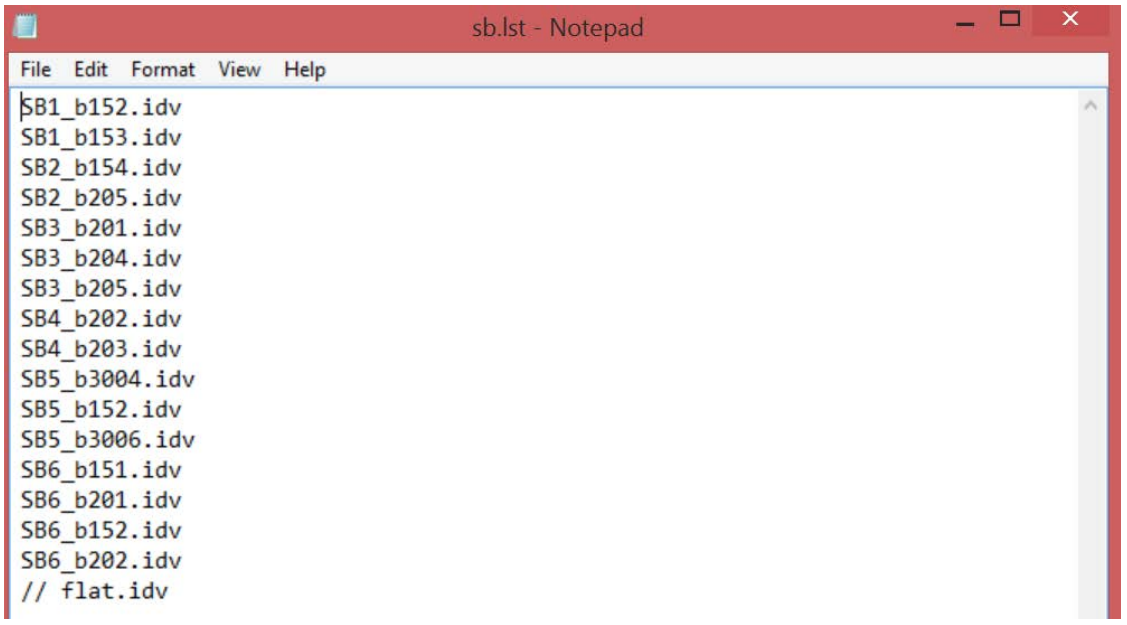
```
savnw_flat33
```

Figure 4.4. xvars Setup for **Parallel Computing of DCAT run**

Note that files ".snp" and ".sav" adopted here are previously generated in the DCAT package by the module "Make_DYR_withPNNLrelays__xx.py"; however, they need to be renamed to be exactly "SAVNW_Finalv33.snp" and "SAVNW_Finalv33_con.sav", respectively.

4.2.1.2 yvars Setup

To set up the yvars, simply establish another ".lst" file with all the ".idv" files, which specify the contingency conditions listed in it. All the ".idv" files listed in the ".lst" file need to be put in the same folder as the ".lst" file. Here we generate a file named "sb.lst" to list all the ".idv" files.



```
SB1_b152.idv
SB1_b153.idv
SB2_b154.idv
SB2_b205.idv
SB3_b201.idv
SB3_b204.idv
SB3_b205.idv
SB4_b202.idv
SB4_b203.idv
SB5_b3004.idv
SB5_b152.idv
SB5_b3006.idv
SB6_b151.idv
SB6_b201.idv
SB6_b152.idv
SB6_b202.idv
// flat.idv
```

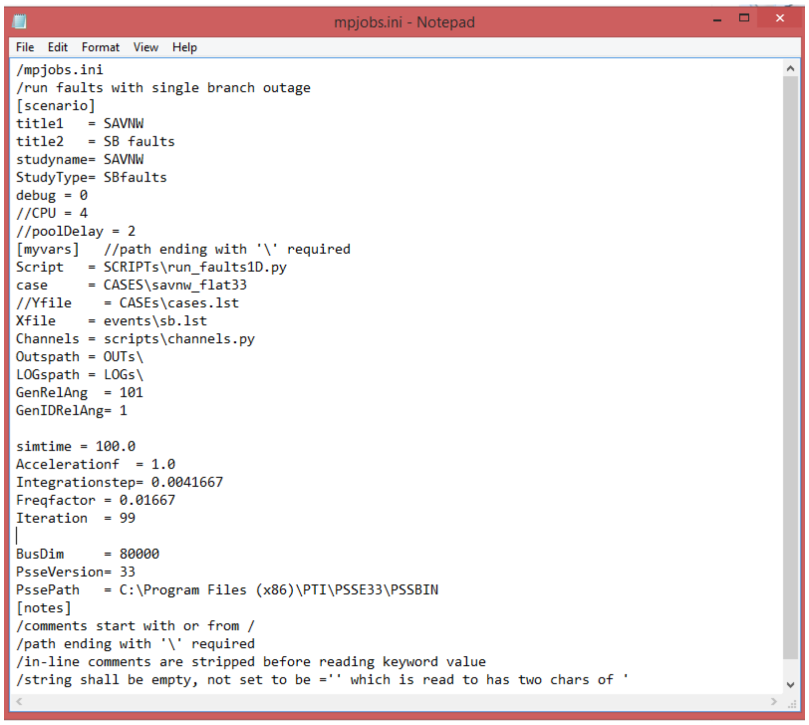
Figure 4.5. yvars Setup for Parallel Computing of DCAT run

Note that the ".idv" file here is the one that can be adopted as the initial contingency file in the previous examples.

4.2.2 Configuration of the “.ini” File

The “.ini” file provides important information for the main function “mpjobs.py” to assign the scenarios and conduct the simulations accordingly. Figure 4.6 shows the configuration on the “mpjobs.ini” file to run the DCAT process:

- “studyname” and “studyType” will be used to form the first parameter of the vector “ZYXvars”.
- “CPU” specifies how many processors to use.
- “poolDelay” specifies the pool delay in second (usually the pool delay needs to be increased when the time of a single run of the “Script” becomes longer).
- “Script” is the name of the main Python script to run PSS®E simulation (here it should be the name of the script to run the main DCAT process).
-
- “Xfile” is the path as well as the “.lst” file to specify the xvars (here it should be “cases.lst”).
- “Yfile” is the path as well as the “.lst” file to specify the yvars (here should be “sb.lst”).
- The parameters within the red rectangle (in Figure 4.3) specify the parameters and supporting files to be used during the computing process. Those parameters will be referred to in the Python module specified by “Script” in the “.ini” file. (Note that when referring to those parameters in the Python module, the name should be all capitalized, and My[‘’] should be added. For example, when referring to “Inlfile” in the Python module specified by “Script”, the name should be modified to My[‘INLFILE’]).



```
/mpjobs.ini
/run faults with single branch outage
[scenario]
title1    = SAVNW
title2    = SB faults
studyname= SAVNW
StudyType= SBfaults
debug = 0
//CPU = 4
//poolDelay = 2
[myvars]   //path ending with '\' required
Script    = SCRIPTs\run_faults1D.py
case      = CASES\savnw_flat33
//Yfile     = CASEs\cases.lst
Xfile     = events\sb.lst
Channels = scripts\channels.py
Outspath = OUTs\
LOGspath = LOGs\
GenRelAng  = 101
GenIDRelAng= 1

simtime = 100.0
Accelerationf  = 1.0
Integrationstep= 0.0041667
Freqfactor = 0.01667
Iteration  = 99

BusDim      = 80000
PsseVersion= 33
PssePath    = C:\Program Files (x86)\PTI\PSSE33\PSSBIN
[notes]
/comments start with or from /
/path ending with '\' required
/in-line comments are stripped before reading keyword value
/string shall be empty, not set to be ='' which is read to has two chars of '
```

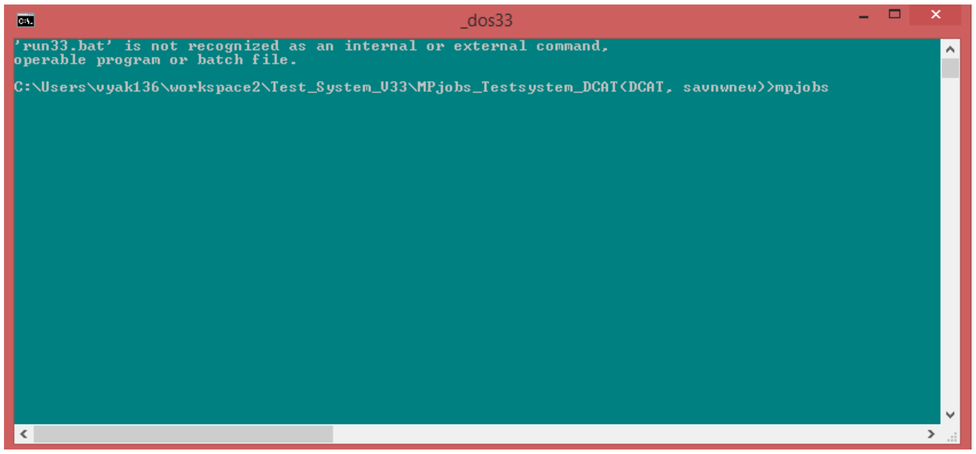
Figure 4.6. Configuration of the “mpjobs.ini” File

4.2.3 Configuration of the Main Python Script to Run PSS®E Simulation

In the DCAT package introduced above, the Python module "MainDCAT.py" is the main script to run the full DCAT process. However, before running this script, several steps need to be completed: prepare ".sav" and ".snp" files, run flat start, etc. Since those steps are not scenario-based and can be done easily and separately, we are not going to include the previous several steps in the "MPjobs" package. Therefore, the module "MainDCAT.py" becomes the only main script to be adopted to run the PSS®E simulation in the "MPjobs" package.

4.3 Run Parallel Computing of DCAT Process

To run parallel computing of DCAT, simply open the "_dos32", "_dos33", or "_dos34" shortcut, type in "mpjobs", as shown in Figure 4.7, and then press "Enter".



```
'run33.bat' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\vyak136\workspace2\Test_System_V33\MPjobs_Testsystem_DCAT(DCAT, savnwnew)>mpjobs
```

Figure 4.7. How to Run the Program

4.4 Some Notes on "MPjobs"

There are several notes regarding running the "MPjobs" package:

- When a self-developed Python file is imported, it should be put in the folder "SCRIPTs", which is the same folder as that of the main function "mpjobs".
- When referring to the parameters specified in the ".ini" file, the name of the parameter should be all capitalized, and My['''] should be added. For example, when referring to "Inlfile" in the ".ini" file, the name should be modified to My['INLFILE'].
- When writing the subroutines in the "MPjobs" environment, some files need to be imported again at the beginning of the subroutine, although they are already imported at the beginning of that Python script.
- It seems that there are some limitations on the number of layers of calling a subroutine from another subroutine.
- No "exit()" function shall be used in the main Python script to run PSS®E simulations.

- The setting of the parameter “poolDelay” is very important, especially when the single run of the main Python script takes a long time. Try to increase the number specified by “poolDelay” if you cannot see that all the output files are generated appropriately.

When modifying the Python script for “MPjobs” to run, it is not recommended to do that directly through Notepad, especially when the modification requires you to add a new line and spaces are needed before the new line. (One can simply copy and paste the script into Eclipse, do the modification, and then copy and paste the modified version back to the original file.)

4.5 Example of Using “Mpjobs”

Here is an example showing how to use the “MPjobs” package to simulate the designated initial contingency. Suppose that the consequences of four contingencies are going to be investigated, and those contingencies are specified in the following “.idv” files: “SB1_b152.idv”, and “SB2_b205.idv”.

Step 1: Check Files and Create Folders as Needed

First, make sure the files needed for running the DCAT process are available. Then create folders to store those files. Here, to be clear, a folder named “TEST” is created in the “MPjobs” folder. Figure 4.8 shows the detailed hierarchy of the new folders created as well as the files in each folder. The folders “LOGs”, “OUTs”, and “SAVs” store the “.log” file, the “.out” file, the “.csv” file (the file to extract the relay action and other important information) as well as the “.raw” file, “.sav” file produced during the simulation process.

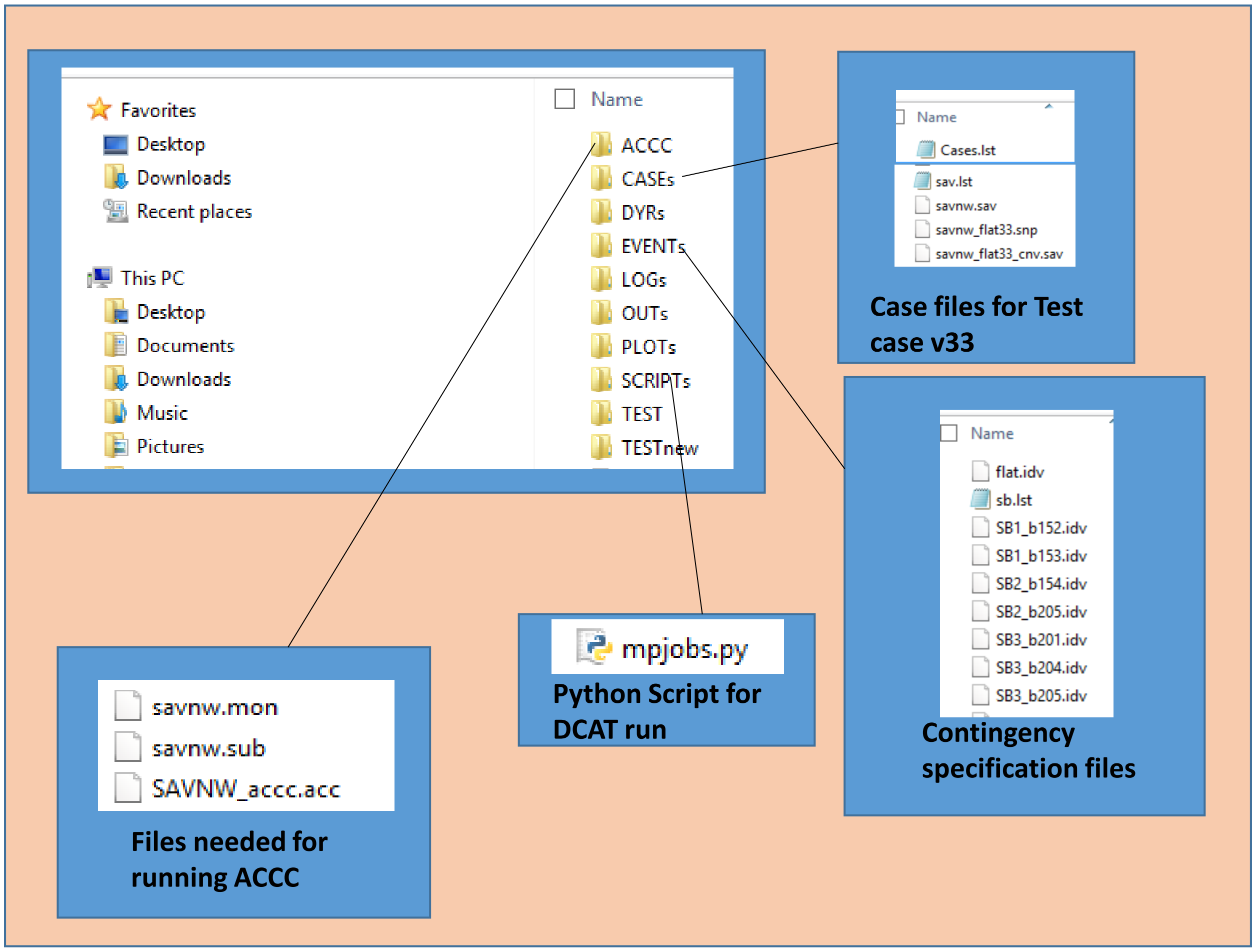


Figure 4.8. **Detailed Hierarchy of New Folders**

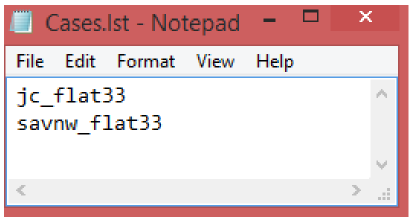
Step 2: Set Up Scenarios and Cases

The file “cases.lst” in the folder “CASEs” sets up the “xvars”. Since the contingency simulation is just going to be conducted on the same base case, just ONE xvar is needed. Therefore, in the “cases.lst”, one just needs to write “SAVNW_Finalv33”, as shown in Figure 4.9.

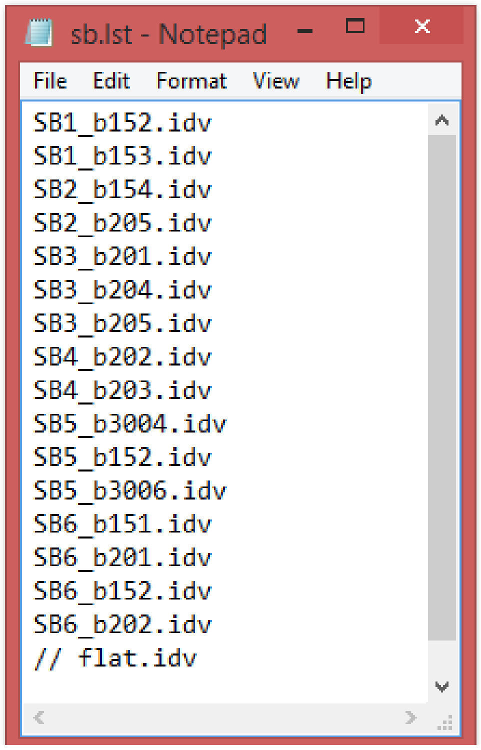
The file “sb.lst” in the folder “events” sets up the “yvars”. Four contingency scenarios are supposed to be investigated, and therefore, the names of the “.idv” files need to be put in the “sb.lst”, as shown in Figure 4.9.

Step 3: Configure “.ini” File

The configuration on the “.ini” file is shown in Figure 4.6. The path as well as the file name of the modified Python module “mpjobs.py” is assigned to the variable “Script”; the path as well as the file name of the “yvars”-specification file “sb.lst” is assigned to the variable “Yfile”; the path as well as the file name of the “xvars”-specification file “cases.lst” is assigned to the variable “Xfile” in the “.ini” file; the ACCC related files are assigned to the variables “Inlfile”, “MySUB” and “MyMON”, respectively.



```
jc_flat33
savnw_flat33
```



```
SB1_b152.idv
SB1_b153.idv
SB2_b154.idv
SB2_b205.idv
SB3_b201.idv
SB3_b204.idv
SB3_b205.idv
SB4_b202.idv
SB4_b203.idv
SB5_b3004.idv
SB5_b152.idv
SB5_b3006.idv
SB6_b151.idv
SB6_b201.idv
SB6_b152.idv
SB6_b202.idv
// flat.idv
```

Figure 4.9. Detailed Configurations on "cases.lst" and "sb.lst"

Step4: Run the Program

After typing in "mpjobs" in the DOS environment (as shown in Figure 4.7), press "Enter" and the program starts to run. The results are stored in the folders "LOGs" and "OUTs".

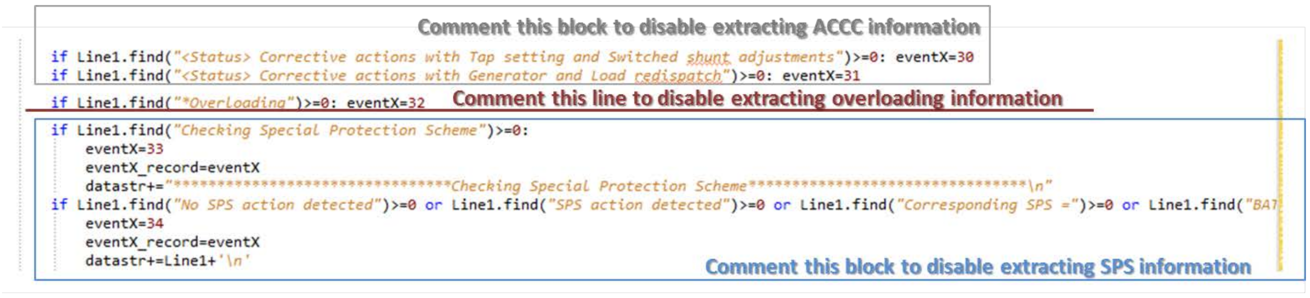
5.0 Introduction of Different Python Modules in the DCAT Package

5.1 “Supportingtools.py” (Supporting File)

This Python module defines many useful subroutines regarding file processing that were created by ERCOT. This module serves as a supporting file and is used by other Python modules. This file does not need to be modified by users during the regular use of this DCAT package.

5.2 “ReadLogFile.py” (Extract Information from “.log” File)

This Python module defines useful subroutines regarding reading the “.log” file generated by PSS®E and extracting useful information such as the relay tripping, ACCC action, SPS, etc. This module also serves as a supporting file and is used by other Python modules. This file does not need to be modified by users during the regular use of this DCAT package. Please refer to Figure 5.1 if ACCC information, SPS information, or overloading information is not needed.



```
if Line1.find("<Status> Corrective actions with Tap setting and Switched shunt adjustments")>=0: eventX=30
if Line1.find("<Status> Corrective actions with Generator and Load redispatch")>=0: eventX=31
if Line1.find("*Overloading")>=0: eventX=32
if Line1.find("Checking Special Protection Scheme")>=0:
    eventX=33
    eventX_record=eventX
    datastr+="********************************Checking Special Protection Scheme********************************\n"
if Line1.find("No SPS action detected")>=0 or Line1.find("SPS action detected")>=0 or Line1.find("Corresponding SPS =")>=0 or Line1.find("BA
    eventX=34
    eventX_record=eventX
    datastr+=Line1+'\n'
```

Comment this block to disable extracting ACCC information
Comment this line to disable extracting overloading information
Comment this block to disable extracting SPS information

Figure 5.1. How to Disable Extracting Some Information

5.3 “Make_DYR_withPNNLrelays_xx.py” (Obtain “.sav” File and “.snp” for Subsequent Simulations)

This Python module loads PSS®E case, makes the necessary changes to the network and converts the case. It basically does the following:

1. Load PSS®E case.
2. Calculate power flow.
3. Make necessary changes to the opened case, including: Transmission System Planning (TSP)changes, static compensator (STATCOM) modeling, SPS models, etc.
4. Make more changes to the opened case, including: 1) modification of the tap-ratio of some transformers, 2) modification of the ratings of some transformers, and 3) modification of the rate C of all branches (both transformer branches and non-transformer branches). (The rate C has been reset to the minimum (130% of rate A, 115% of rate B), and the rate C will be used to check the overloading when needed in the subsequent simulation).
5. Convert the case using desired generator and load models.
6. Save the case.

Note: If any changes related to the network are needed, they should be specified in this Python file.

7. This Python module also reads in the converted ".sav" file, adds dynamic data saved in ".dyr" files and creates ".snp" files. It basically does the following:
   I. Load the converted ".sav" file.
   II. Process dynamic files to load dynamic models and link user-defined models. It is important to note that if any changes are made to the user-defined model, which is the part before "dyre_new", a new compilation might be needed.
   III. Make necessary changes regarding the dynamic models and parameters.
   IV. Save the case to the ".snp" file.

Note: If any changes regarding the dynamic models or data are needed, they should be specified in this Python file.

5.4 "RunFlatStart.py" (Run Flat Start)

This Python module runs a flat start on the given case, and it basically does the following:

1. Load the converted ".sav" file and the dynamic file ".snp".
2. Add all the DLLs.
3. Set channels through another Python script, "channels_flat.py".
4. Run flat start.

The default time is 10 s. If that time duration needs to be modified, you can simply modify the parameter "Time_parameter" in that module.

5.5 "RunTestFault.py" (Dynamic Simulation Given a Certain Contingency)

This Python module runs a dynamic simulation in a specified contingency. It basically does the following:

1. Load the converted ".sav" file and the dynamic file ".snp".
2. Add all the DLLs.
3. Set channels through another Python script "channels_flat.py".
4. Run dynamic simulation.

The contingency is specified in an ".idv" file. The default total simulation time is 20 s. If that time duration needs to be modified, you can simply modify the parameter "Time_parameter" in that module.

5.6 “MainDCAT.py” (Main DCAT Process Given a Certain Contingency)

This is the main module of the DCAT package. It is able to simulate the potential cascading events in a given contingency. The detailed introduction of this module is provided in section 6.0.

5.7 “Configuration File”

The configuration file provides the necessary settings for the Python modules “RunFlatStart.py”, “RunTestFault.py”, and “MainDCAT.py”. The detailed introduction on each parameter is both commented in the configuration file (“RunFlatStart_Config.ini”/ “Config.ini”) and provided in sections 3.3 to 3.5. Please note:

1. There are three major parts in the configuration file, which provide the settings for the Python modules “MainDCAT.py”, “RunTestFault.py”, and “RunFlatStart.py”, respectively: “Settings for MainDCAT”, “Settings for RunTestFault”, and “Settings for RunFlatStart”;
2. The comment in this configuration file starts with “;”
3. The symbol “:” separates the parameter name and the value of that parameter;
4. If the file is not located in the same folder as the module “MainDCAT.py” is, the path of the file should also be provided.

6.0 Introduction of “MainDCAT.py”

“MainDCAT” is the main Python module in the DCAT package, and it consists of several subroutines and a main function. The description of some major subroutines and illustration of the main flow are given in this section.

6.1 Main Flow of “MainDCAT.py”

The main flow of the Python module “MainDCAT” is illustrated in Figure 6.1

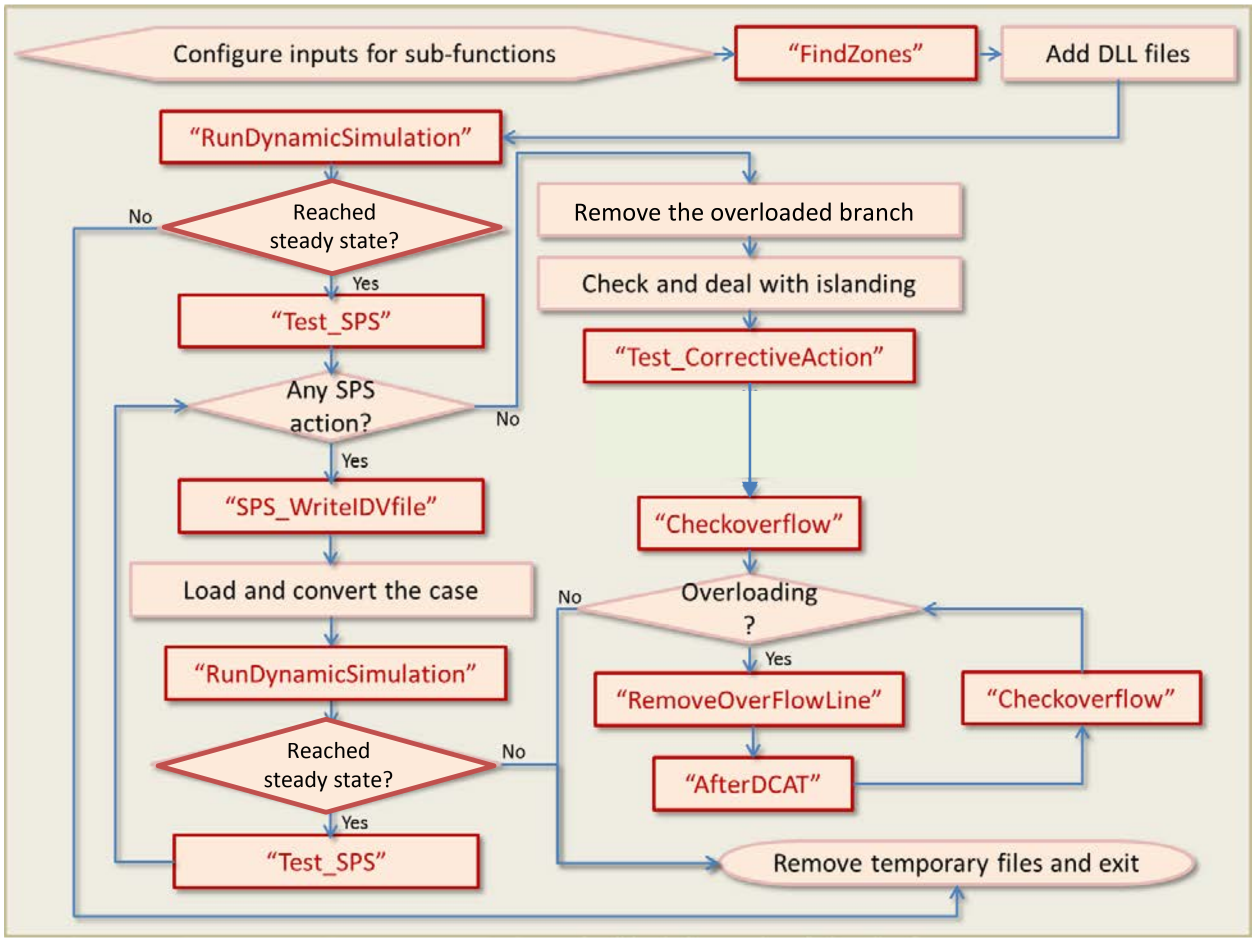


Figure 6.1. Illustration of Main Flow of “MainDCAT”

6.2 Subroutines in “MainDCAT.py”

6.2.1 “RunDynamicSimulation”

This subroutine runs a dynamic simulation and returns the number of iterations needed for the dynamic simulation to reach a new quasi-steady state.

Input(s):

- the names of the log file and prompt log file to be produced;
- the ".sav" and ".snp" file of the case, the ".idv" file defining a contingency;
- the name of the ".out" file to be generated;
- the name of the ".raw" file returned after the dynamic simulation.

Output(s):

- the number of iterations needed for the dynamic simulation to reach a new quasi-steady state.

This subroutine is able to judge whether the dynamic simulation has reached a steady state or not, and if not, another 5 seconds of dynamic simulation will be performed until the dynamic simulation reaches a new steady state or the number of iterations, "MaxNumIter", is reached. The main flow of this subroutine is shown in Figure 6.2 ("MaxNumIter" is the "nMax" in Figure 6.2).

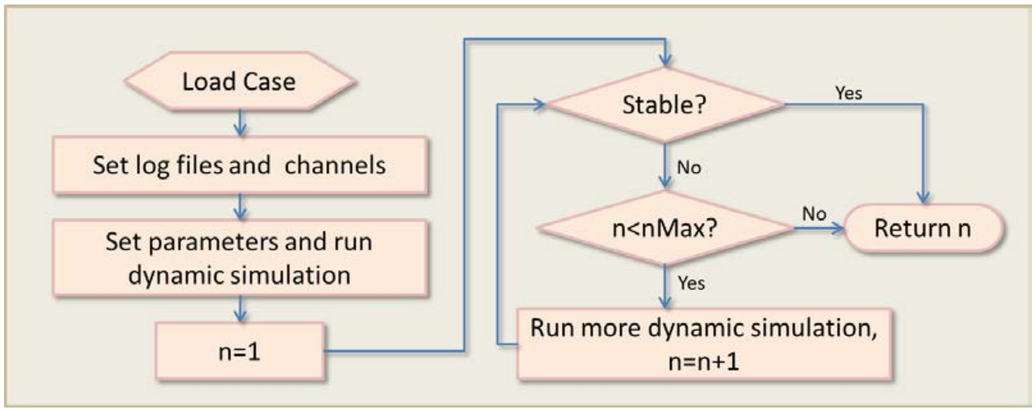

Load Case
Set log files and channels
Set parameters and run dynamic simulation
n=1
Stable?
Yes
No
n<nMax?
No
Return n
Yes
Run more dynamic simulation, n=n+1

Figure 6.2. Illustration of Main Flow of "RunDynamicSimulation"

6.2.2 "IsDisconnected"

This subroutine detects whether all the buses have been disconnected from the system. This may happen during some extreme conditions.

Input(s):

- The name of the ".raw" file to be checked.

Output(s):

- A Boolean value to indicate whether all buses are disconnected or not.

6.2.3 "Check_NDM_Island"

This subroutine checks the log file to see whether there is any island without dispatchable generators, and it is called when inertial power flow fails to assign new slack buses. This subroutine reads the log file and returns the list of the islands that do not have dispatchable generators.

Input(s):

- The name of the ".log" file to be checked.

Output(s):

- A list of islands without dispatchable machines.

6.2.4 "Check0serviceBus"

This subroutine checks the log file to see whether there is any bus to which no in-service branches are connected, and it is called when inertial power flow fails to assign new slack buses. This subroutine reads the log file and returns the list of buses that do not have in-service branches.

Input(s):

- The name of the ".log" file to be checked.

Output(s):

- A list of buses that do not have in-service branches.

6.2.5 "Test_SPS"

This subroutine is called to check the SPS after the dynamic simulation.

Input(s):

- logfile: the name of the log file generated when checking the SPS;
- rawfile: the name of the ".raw" file in which the SPS needs to be checked;
- inlfile: the ".inl" needed for inertial power flow;
- returnfile: the name of the ".sav" file to be saved after finishing the SPS checking;
- spsfile: the ".idv" file that will be generated when SPS action is detected;
- logfile_total: the log file that includes all the information during the full run of the DCAT.

Note that an ".idv" file will only be generated in the situation when any SPS action is detected. Besides, the content within the "logfile" needs to be written into the "logfile_total", since the "logfile" is just a temporary file and will be deleted eventually. The main flow of this subroutine is illustrated in Figure 6.3.

Output(s):

- An ".idv" file if any SPS action is detected.

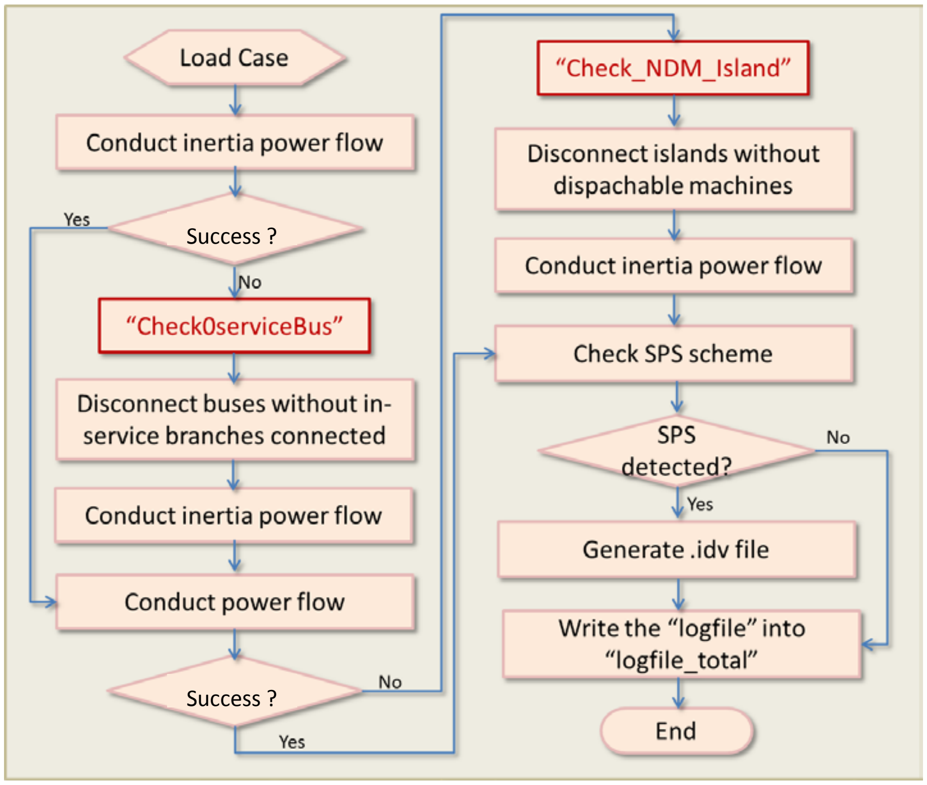

Load Case
Conduct inertia power flow
Success ?
Yes
No
"Check0serviceBus"
Disconnect buses without in-service branches connected
Conduct inertia power flow
Conduct power flow
Success ?
No
Yes
"Check_NDM_Island"
Disconnect islands without dispachable machines
Conduct inertia power flow
Check SPS scheme
SPS detected?
No
Yes
Generate .idv file
Write the "logfile" into "logfile_total"
End

Figure 6.3. Illustration of Main Flow of "Test_SPS"

6.2.6 "SPS_WriteIDVfile"

This subroutine is called when any SPS action is detected. It reads the ".idv" file generated when checking SPS and produces a new ".idv" file to be used in the new dynamic simulation.

Input(s):

- The name of the ".idv" file generated during the SPS checking process.
- The name of the new ".idv" file to be generated for the following dynamic simulation.

Output(s):

- A new ".idv" file to conduct SPS action for the following dynamic simulation.

6.2.7 "FindVirLine"

This subroutine is used to find a virtual branch in the opened case in order to run the AC contingency analysis. What this function basically does is to search for an existing non-transformer branch and then return the information of a new branch in parallel with that branch.

Input(s):

- This subroutine is called after a particular PSS®E case is opened; no other inputs are needed.

Output(s):

- “From” bus of the virtual branch;
- “To” bus of the virtual branch;
- Branch ID of the virtual branch.

6.2.8 “ModifyConFile”

This subroutine is called to generate a “.con” file, which is to be adopted in the ACCC simulation, according to the information of the branch that is going to be switched off during the AC contingency analysis.

Input(s):

- The name of the “.con” file to be generated;
- “From” bus of the branch that is going to be switched off;
- “To” bus of the branch that is going to be switched off;
- Branch ID of the branch that is going to be switched off.

Output(s):

- The “.con” file generated to switch off a designated branch during the ACCC simulation.

6.2.9 “Test_CorrectiveAction”

This subroutine runs the AC contingency analysis after the dynamic simulation.

Input(s):

- The “.sav” file of the network to be analyzed;
- The name of the log file;
- The name of the “.con” file to be generated and adopted;
- The name of the “.dfx” file to be generated and adopted;
- The name of the “.sav” file to be saved after the ACCC;
- The name of the “.raw” file to be saved after the ACCC.

Output(s):

- A label indicating whether ACCC succeeds or not.

The main flow of this sub-function is presented in Figure 6.4.

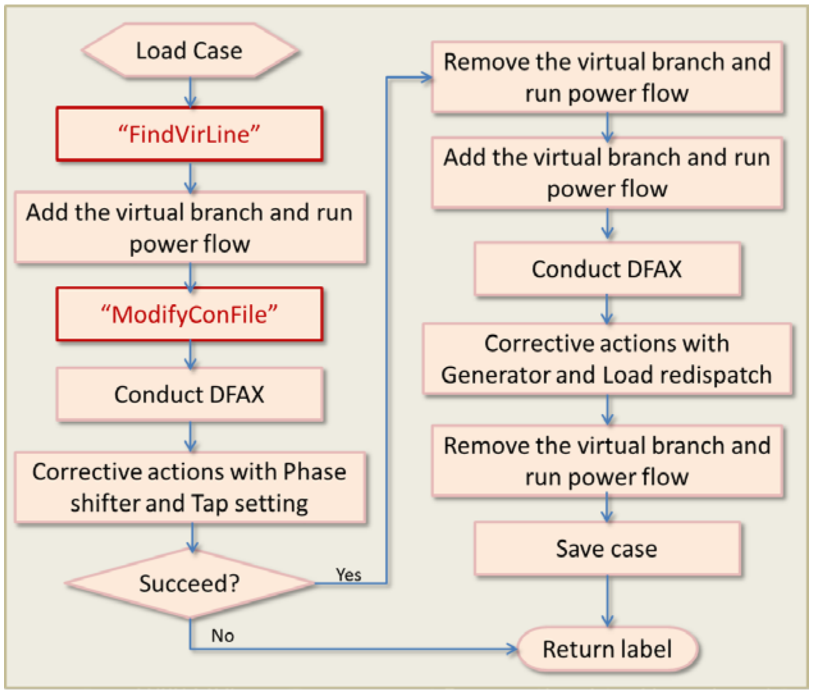

Load Case
"FindVirLine"
Add the virtual branch and run power flow
"ModifyConFile"
Conduct DFAX
Corrective actions with Phase shifter and Tap setting
Succeed?
Yes
No
Remove the virtual branch and run power flow
Add the virtual branch and run power flow
Conduct DFAX
Corrective actions with Generator and Load redispatch
Remove the virtual branch and run power flow
Save case
Return label

Figure 6.4. Illustration of Main Flow of "Test_CorrectiveAction"

6.2.10 "FindZones"

This subroutine reads the ".sub" file and returns a list of zone numbers listed in the ".sub" file. If the ".sub" file cannot be found, the subroutine will return a default list of zone numbers.

Input(s):

- The name of the ".sub" file.

Output(s):

- A list of the zone numbers defined in the ".sub" file, or a default list of numbers if a ".sub" file cannot be found.

6.2.11 "Checkoverflow"

This subroutine checks the overloading on the branches within the subsystem that are generated according to a list of zone numbers.

Input(s):

- A list of zone numbers generated by the subroutine "FindZones".

Output(s):

- A label indicating: 1) no overloading; 2) maximum overloading exists in a non-transformer branch or a two-winding transformer branch; 3) maximum overloading exists in a three-winding transformer branch;
- The information of the branch with the maximum overloading.

6.2.12 “RemoveOverflowLine”

This subroutine is called when the subroutine “Checkoverflow” finds and returns the information on the branch with maximum overloading. It produces an “.idv” file to trip the corresponding overloading branch for the later dynamic simulation.

Input(s):

- A label indicating: 1) no overloading; 2) maximum overloading exists in a non-transformer branch or a two-winding transformer branch; 3) maximum overloading exists in a three-winding transformer branch;
- The information of the branch with maximum overloading;
- The name of the “.idv” file to be generated.

Output(s):

- An “.idv” file to trip the corresponding overloading branch for the later dynamic simulation.

6.2.13 “AfterDCAT”

This subroutine basically does the following: 1) runs dynamic simulation to switch off the overloading branch; 2) keeps checking SPS and running dynamic simulation if any SPS action is detected; 3) runs ACCC simulation; 4) checks overloading. The process will stop if any of the following scenarios occurs: 1) the dynamic simulation does not reach a new quasi-steady state; 2) all buses are disconnected; 3) the dynamic simulation cannot initialize properly. This subroutine is like one iteration of the DCAT, and it is called after the first iteration of the DCAT with the initiating contingency.

Input(s):

- The name of the log file to be generated;
- The name of the prompt log file;
- The “.sav” file to be used in the dynamic simulation;
- The “.snp” file to be used in the dynamic simulation;
- The “.idv” file specifying the contingency condition for the dynamic simulation;
- The name of the “.out” file to be generated during the dynamic simulation;
- The name of the “.sav” file to be saved after the dynamic simulation;
- The number indicating the maximum iterations in the dynamic simulation;
- The name of the log file temporarily generated when checking the SPS;
- The .inl file needed for the inertial power flow;
- The name of the “.sav” file after checking the SPS;
- The name of the “.idv” file to be generated if any SPS action is detected;
- The name of the “idv” file generated that corresponds to the SPS action detected for the later dynamic simulation;
- The name of the log file during the ACCC;

- The name of the ".sav" file after removing the overloaded branch and conducting island checking, and this file is the input for ACCC;
- The name of the ".con" file to be generated for ACCC;
- The name of the ".dfx" file to be generated for ACCC;
- The name of the ".sav" file to be saved after ACCC;
- The name of the ".raw" file to be saved after ACCC;
- The label indicating whether the maximum loading is within a non-transformer branch, a two-winding transformer branch or a three-winding transformer branch;
- The branch information.

Output(s):

- A label describing whether the dynamic simulation has reached a steady state;
- A label describing whether all the buses are disconnected;
- A label describing whether the ACCC has succeeded or not.

The main flow of this subroutine is shown in Figure 6.5.

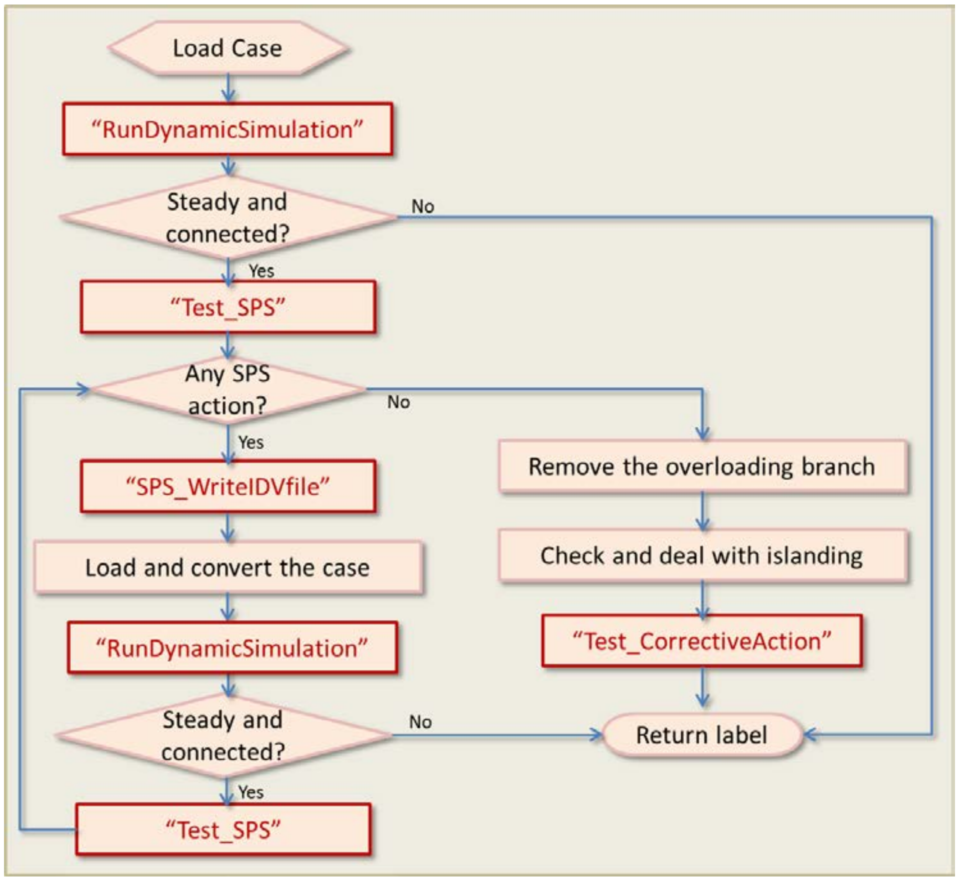


Figure 6.5. Illustration of Main Flow of "AfterDCAT"

7.0 References

[1] NA Samaan , JE Dagle, YV Makarov, R Diao, MR Vallem, TB Nguyen, LE Miller, BG Vyakaranam, S Wang, FK Tuffner, and , MA Pai, “Dynamic Contingency Analysis Tool – Phase 1”, PNNL-24843, Pacific Northwest National Laboratory, Richland, WA, 2015. [Online.] Available: http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-24843.pdf

[2] “Application Program Interface (API),” Siemens Industry, Inc., October 2013.

[3] José Conto, “MPjobs – a tool to run PSSe scripts in parallel,” ERCOT, 2015.

Appendix A

Modeling of Protection in Dynamic Simulations Using Generic Relay Models and Settings

Appendix A

Modeling of Protection in Dynamic Simulation Using Generic Relay Models and Settings

NA Samaan , JE Dagle, YV Makarov, R Diao, MR Vallem, TB Nguyen, LE Miller, BG Vyakaranam, and FK Tuffner
Pacific Northwest National Laboratory
Richland, WA, 99352
nader.samaan@pnnl.gov

M.A. Pai
University of Illinois
Urbana-Champaign, IL

Jose Conto and Sun Wook Kang
ERCOT
Taylor, TX, 76574

***Abstract*—This paper shows how generic protection relay models available in planning tools can be augmented with settings that are based on NERC standards or best engineering practice. Selected generic relay models in Siemens PSS®E have been used in dynamic simulations in the proposed approach. Undervoltage, overvoltage, underfrequency, and overfrequency relays have been modeled for each generating unit. Distance-relay protection was modeled for transmission system protection. Two types of load-shedding schemes were modeled: underfrequency (frequency-responsive non-firm load shedding) and underfrequency and undervoltage firm load shedding. Several case studies are given to show the impact of protection devices on dynamic simulations. This is useful for simulating cascading outages.**

***Index Terms*— cascading failures, dynamic simulation, planning models, protection system, relays.**

I. Introduction

Protection systems in modern power networks have been identified by the North American Electric Reliability Corporation (NERC) as a critical reliability asset. After the 2003 North America blackout, based on the U.S.-Canada Power System Outage Task Force report [1], the NERC stated that one of the major causes of this large-scale blackout was overly conservative relay settings, combined with cascading relay operation. Misoperation of protection systems or incorrect settings can contribute to the spread of blackouts, e.g., overreach of Zone 3 protection coverage in transmission line distance relays. Better understanding of protection schemes, their sequence of operation and coordination, careful review of settings, and proper design changes can minimize the impact of disturbances.

The lack of wide-area consolidated dynamic models with protection relay models incorporated is a major challenge for performing power system analysis This includes common analyses, such as model validation, simulating grid disturbances, performing sound cascading-outage analysis, and developing Remedial Action Schemes (RAS)/Special Protection Systems (SPS). The current practice in dynamic simulations is to mimic protection actions in dynamic simulations by assuming that the fault will be cleared, and identifying the elements that will be tripped due to this fault with a certain time delay after fault inception. In addition, some grid operators model underfrequency and undervoltage load-shedding relays in their planning models.

Commercial software tools for large-scale power system steady-state and dynamic simulations, such as GE PSLF, Siemens PSS®E (PSS/E), Powertech TSAT^{TM}, and PowerWorld Simulator, allow inclusion of some generic protection scheme elements, but this capability is not completely adequate or usually employed by utility planning engineers. Specific software packages such as CAPE and ASPEN, designed for coordinating protection relay settings, employ a totally different set of models and simulation approaches with much smaller time steps. These tools are typically used by the protection engineers of the generation, transmission, and distribution asset owners.

Recent research efforts have focused on linking the dynamic simulations between CAPE and PSS/E to provide a more accurate simulation result for a large-scale system [2]. There have been ongoing efforts in the Western Electricity Coordinating Council (WECC) to develop generic relay models for dynamic simulations such as the development of the Generic Distance Relay Model [3].

In this paper, an approach for adding generic protection system models to planning tools and performing more realistic PSS/E dynamic simulations is described. The generic protection models available in planning tools are augmented with settings that are based on engineering experience and knowledge of general principles and solutions of protection systems. Selected generic relay models in PSS/E have been used in dynamic simulations as follows. Undervoltage, overvoltage, underfrequency, and overfrequency relays have been modeled for each generating unit. Out-of-step protection has been implemented through a user-written model that is applied only to synchronous machines. Distance-relay protection generic models have been used for the transmission system. Two types of load shedding schemes were modeled: underfrequency (frequency-responsive non-firm load shedding) and underfrequency and undervoltage firm load shedding. The proposed generic relay settings can later be improved based on communications between planning and protection engineers.

The remainder of the paper is organized as follows. Section II describes generator protection modeling. Section III discusses relay models used for transmission protection. Section IV explains modeling of load-shedding relays. In section V, case study examples are given to show the impact of protection modeling on dynamic simulations. Finally, some concluding remarks and topics for future investigation are provided in Section VI.

II. Generation Protection Modeling

A key aspect to dynamic simulations of power systems is properly modeling the response of generators to various grid events. Adequately representing the protective devices on the generator helps capture their operational and planning impacts to the system. This section covers two typical protection schemes for generators and the implementation details inside PSS/E.

A. Voltage and Frequency Based Generation Protection

Protection modeling of generation units uses under-voltage/over-voltage generator disconnection relay model, VTGTPA, and under-frequency/over-frequency generator disconnection relay model, FRQTPA, from the PSS/E model library. These relay models are used to protect a single generator, rather than to disconnect an entire generator bus. The settings for the tripping of these relays are taken from NERC Standard PRC-024-1, "Generator Frequency and Voltage Protective Relay Settings" [4], which will become effective in 2016. The over/undervoltage settings used are shown in Table I. The over/underfrequency settings used are shown in Table II.

Table I. Under/Overvoltage Settings for Generator Protection

High-Voltage Ride-Through Duration		**Low-Voltage Ride-Through Duration**	
Voltage (pu)	**Time (s)**	**Voltage (pu)**	**Time (s)**
≥1.200	Instantaneous trip	<0.45	0.15
≥1.175	0.20	<0.65	0.30
≥1.15	0.50	<0.75	2.00
≥1.10	1.00	<0.90	3.00

Table II. Under/Overfrequency Settings for Generator Protection

High-Frequency Duration		**Low-Frequency Duration**	
Frequency (Hz)	**Time (s)**	**Frequency (Hz)**	**Time (s)**
≥61.8	Instantaneous trip	≤57.5	Instantaneous trip
≥61.6	30	≤58.0	2
≥60.6	540	≤58.4	30
<60.6	Continuous operation	≤59.4	540
		>59.4	Continuous operation

All the PSS/E generator protection relay models include the following two parameters:

- *TP*, which is the relay pickup time in seconds. This parameter is set to a minimum of 50 μs, which is taken from the Schweitzer Engineering Laboratories datasheet, "SEL-700G Family of Generator and Intertie Protection Relays" [5]. The parameter *TP* is also used to implement different time-delayed settings.
- *TB*, which is the breaker time in seconds is set to 83 ms, which is taken from "IEEE Standard for AC High-Voltage Circuit Breakers Rated on a Symmetrical Current Basis - Preferred Ratings and Related Required Capabilities for Voltages Above 1000 V" [6].

B. Generator Out-of-Step Protection

A user-written model, Generator Scan Angle (GNSCNANG), has been developed by a Siemens PSS/E team for this study. This model scans all rotor angles at each time step during the dynamic simulation and trips generators that have rotor angles advanced across a specified threshold compared to a chosen reference angle. The operation of this relay model mimics the operation of out-of-step relay protection.

The reference angle is chosen as the center of inertia (COI) angle. This user-written model scans only synchronous generators. In this work, the threshold is chosen to be 180 degrees, and the reference angle is defined as

$$\delta_{COI} = \frac{1}{H_T}\sum_{i=1}^{N} H_i\delta_i;\; H_T = \sum_{j=1}^{N} H_j;\; \delta^i_{COI} = \delta_i - \delta_{COI} \qquad (1)$$

where

N = total number of synchronous machines considered
H_i = moment of inertia of the j^{th} machine
H_T = system inertia
δ_i = rotor angle of the j^{th} machine
δ_{COI} = reference angle in the COI reference frame
δ^i_{COI} = relative rotor angle of the i^{th} machine in the COI reference frame

As soon as the relative rotor angle in the COI reference frame is greater than the threshold value, the generator should trip.

III. Distance Relay Protection for Transmission Lines

For transmission line protection systems, the PSS/E distance relay model DISTR1 has been used. Given that typical dynamic simulations only run for 30 seconds, overcurrent relay models, which typically take longer time to operate, have not been considered.

A. Distance Relay Placement

Typically, transmission line breaker locations are not available in the bus-branch planning models; rather, they are available in grid models used in protection software packages and node-breaker operation models such as the Common Information Models (CIM). In our proposed approach, Category B contingency lists based on their definition in the old NERC reliability standards (TPL-003-0b and TPL-004-0a) [7] have been used to determine breaker locations for the placement of protection within the transmission network. A more accurate approach for determining placement of protective devices is to extract the information from CIM database.

For transmission lines at 230 kV and above, two relays are assumed, one at each end, to fully protect a line. For lower voltage lines, breaker placement is based on information available in Category B contingency definitions which identify transmission circuits tripping due to the same fault, e.g., upper part of Fig.1 shows a structure consisting of four branches in series. Distance relays are assumed at each end of the structure. In the case of n number of lines, the impedance settings for the two relays are set to treat the length of the structure as the line length: $Z_{relay} = Z_{line\_1} + \ldots\ldots + Z_{line\_n}$. There could be some series branches with a single lateral branch as shown in the lower part of Fig. 1. In this case, three distance relays are assumed.

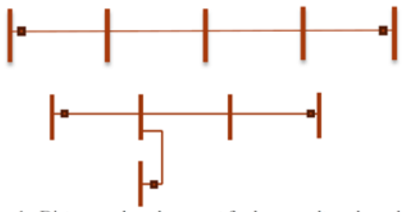


Fig. 1. Distance relay placement for lower voltage branches

B. Distance Relay Model Settings

It is fairly typical to set Zone 1 for distance relays at 85–90% of the line length, Zone 2 at 120–150% of the line length, and Zone 3 at 150% of the next line. Operation of Zone 2 of the distance relay for the line must be coordinated with the Zone 1 setting of the next line such that Zone 1 of the next line must operate before Zone 2 of the first line does. This coordination delay for Zone 2 is usually on the order of 0.3 s. Similarly, operation of Zone 3 for the line must coordinate in time and distance with Zone 2 of the next line. The operating time of Zone 3 is usually on the order of 1 s [3].

In the proposed approach, generic Zone 1 and Zone 2 settings that are similar across all distance relays have been used, with settings based on the values of X and R of the corresponding line that could be obtained from the branch table in the planning model. The Zone 3 protection approach is beyond the scope of this paper.

IV. Modeling of Load-Shedding Relays

Two types of load shedding schemes are modeled: underfrequency (frequency-responsive non-firm load shedding) and underfrequency and undervoltage firm load shedding.

Underfrequency load-shedding relays drop load on a predetermined schedule to balance load and generation during contingencies. Typically there is a sequence of shedding increments of load if frequency continues to drop [8]. Fig. 2 shows a sequence of frequency load-shedding points on the frequency (horizontal) axis; time is on the vertical axis. Load-shedding relay settings are based on the guidance of NERC Standard PRC-006-NPCC-1, "Automatic Underfrequency Load Shedding."

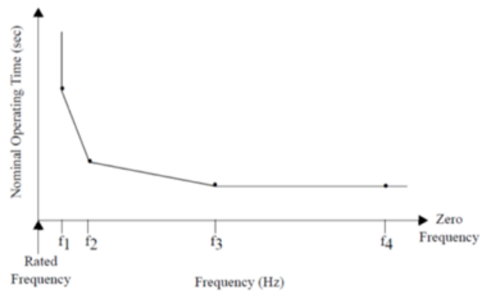


Fig. 2. Time versus frequency curve showing load-shedding points along the frequency axis

In addition, undervoltage load-shedding relays drop load on a predetermined schedule if voltage drops below a certain value. That is typically done in a number of steps with different portions of load to be shed and corresponding thresholds for different steps. Load-shedding relay settings are based on NERC Standard PRC-022-1, "Under-Voltage Load Shedding Program Performance.

V. Case Studies

Generic protection relay models explained in the previous section are added to the test case "savnw.sav" available in the PSS/E example library. A one-line diagram of the test case is shown in Fig. 3. The following case studies are given to show the impact of protection modeling on dynamic simulations.

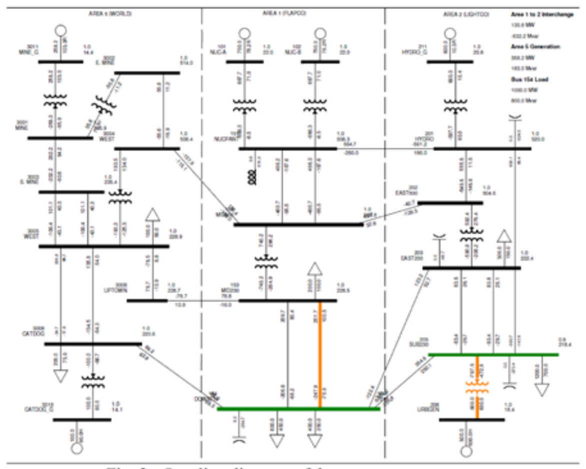


Fig. 3. One-line diagram of the test system

A. Test 1: Not a Close-In Fault in Pilot Scheme Line – Using Fictitious Node

To model a fault in PSS/E at any location in a transmission line other than the two line ends, a fictitious node needs to be added. In this test, to model a fault in the line connecting buses 151 and 152 that is located at a distance of 10% of total line length from bus 151, a new fictitious node (151152) is added between buses 151 and 152. Fig. 4 shows the location of the fictitious bus. Distance relays then need to be associated with the two branches newly created by the fictitious bus addition. That is, one branch is from the near end to the fictitious bus, and the other is from the remote end to the fictitious bus.

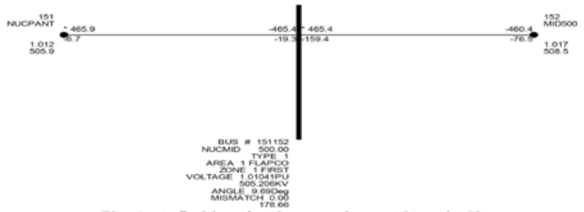


Fig. 4. A fictitious bus between buses 151 and 152

The bus fault is introduced at the fictitious bus (151152) at $t = 5$ s and simulation runs until dynamic simulation reaches a steady state. In this test, dynamic simulation reaches a steady state at $t = 16$ s. The following is the sequence of relay tripping events:

1) Distance relay (DISTR1) at Circuit 1 from 151 to 151152 is activated as Zone 1 and its timer starts at $t = 5$ s.

2) Distance relay (DISTR1) at Circuit 1 from 152 to 151152 is activated as Zone 2 and its timer starts at $t = 5$ s.

3) The Zone 1 timer times out at $t = 5.017$ s; the self-trip breaker timer and also transfer trip and breaker trip timers start at the same time.

4) Circuit 1 from 151 to 151152 trips at $t = 5.05$ s and the transfer trip timer also times out at the same time. In this case, the Zone 1 relay accelerates the other relay, and as a result, the other end (Circuit 1 from 152 to 151152) trips at the same time ($t = 5.05$ s), and soon thereafter the two voltages start to recover.

The channel plot in Fig. 5 shows that the voltage at bus 151 collapses more than the bus 152 voltage. This indicates that the fault is closer to bus 151.

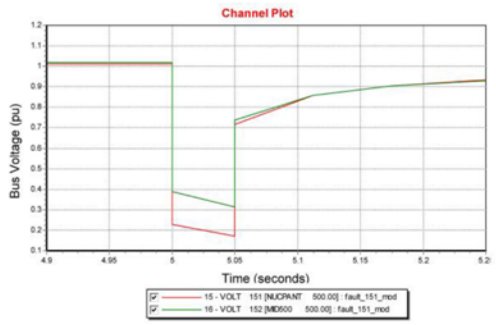


Fig. 5. Voltage plots of the terminal buses of the faulted line for Test 1

B. Test 2: Not a Close-In Fault in Step Distance Line – Using Fictitious Node

This simulation uses the same procedure and files that were used in Test 1 except that the transfer trip capability of DISTR1 is assumed to have failed. Each end will trip according to the Zone 1 or Zone 2 delays where appropriate. The bus fault is introduced at the fictitious bus (151152) at $t = 5$ s and simulation runs until dynamic simulation reaches a steady state. In this test, dynamic simulation reaches a steady state at $t = 16$ s. The following is the sequence of relay tripping events:

1) Distance relay (DISTR1) at Circuit 1 from 151 to 151152 is activated as Zone 1 and its timer starts at $t = 5$ s.

2) Distance relay (DISTR1) at Circuit 1 from 152 to 151152 is activated as Zone 2 and its timer starts at $t = 5$ s.

3) The Zone 1 timer times out at $t = 5.017$ s; the self-trip breaker timer and breaker timer start at the same time.

4) Circuit 1 from 151 to 151152 trips at $t = 5.05$ s.

5) Circuit 1 from 152 to 151152 trips as a Zone 2 fault at t = 5.333 s, and the channel plot (Fig. 5) shows the two voltages start to recover after tripping both ends of the branch.

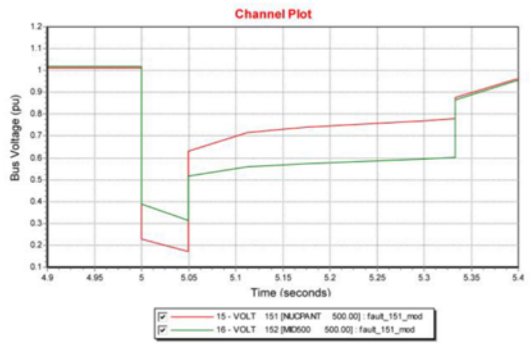


Fig. 6. Voltage plots of the terminal buses of the faulted line for Test 2

C. Test 3: Bus Fault

In Test 3, a fault is applied at bus 201 at $t = 5$ s and the fault is cleared after 10 cycles. The simulation runs until dynamic simulation reaches a steady state at $t = 75$ s. Table III shows a summary of the tripping events. Simulation result plots are shown in Figs. 7 and 8.

TABLE III. Relay Trips Summary of Test 3

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	
DISTR1	5.1	201	202	1	
VTGTPA	TimeOut	Bus	BusName	BuskV	Pgen (MW)
VTGTPA	5.237	211	HYDRO_G	20	600

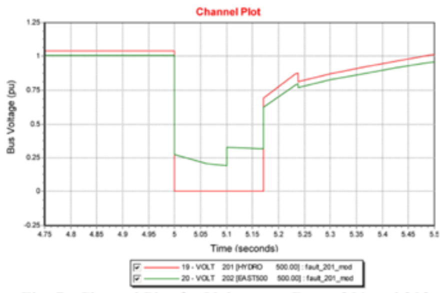


Fig. 7. Channel Plot for Voltages at Buses 201 and 202

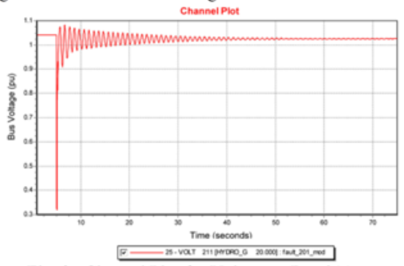


Fig. 8. Channel Plot for Voltage at Bus 211

D. Test 4: Bus Fault Leads to Blackout

In this dynamic simulation, a fault is applied at bus 151 at $t = 5$ s; the fault is applied for 12 cycles and then cleared. A significant number of undervoltage and underfrequency generator relays were tripped due to this fault, which leads to system blackout. The network did not converge after $t = 6.3708$ s. A total of seven relays are activated during this dynamic simulation; Table IV shows a summary. Fig. 9 shows the speed of the selected machines that have tripped.

TABLE IV. Sequence of Relay Trippings in Test 4

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	
DISTR1	5.1	151	152	1	
DISTR1	5.1	151	152	2	
VTGTPA	TimeOut	Bus	BusName	BuskV	Pgen (MW)
VTGTPA	5.237	101	NUC-A	21.6	750
VTGTPA	5.237	102	NUC-B	21.6	750
FRQTPA	TimeOut	Bus	BusName	BuskV	Pgen
FRQTPA	6.362	3018	CATDOG	13.8	100
FRQTPA	6.371	206	URBGEN	18	800
FRQTPA	6.371	3011	MINE_G	13.8	258.66
FRQTPA	6.383	211	HYDRO	20	600

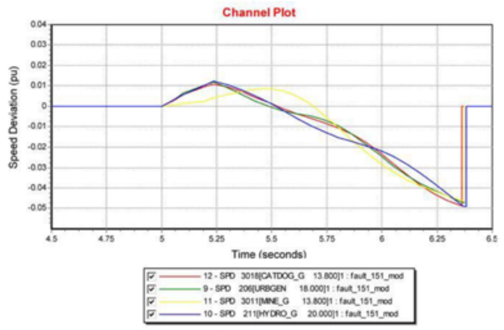


Fig. 9. Channel Plots for Speeds of Selected Machines

VI. CONCLUSION

The lack of integrated dynamic models with protection relays is a major challenge for performing accurate dynamic simulations, especially to understand the impact of protection misoperation. This paper shows main generic relay models that can be added to planning models to include protection system actions. The newly issued NERC Standard PRC-024-1 will make the under/overfrequency and under/overvoltage relay settings universal for all generating units. Grid operators should add these relay models to their dynamic models. In addition, modeling of out-of-step protection for generating units is important. For transmission protection, the modeling of distance relays with correct settings could be a challenge, because this information is typically set by transmission owners and not available to grid operators. These efforts are the starting point for developing comprehensive approaches that consolidate protection models in dynamic simulations that can be easily used by planning engineers.

VII. ACKNOWLEDGMENT

This study is funded by the Office of Electricity Delivery and Energy Reliability at the U.S. Department of Energy. The project team wants to especially thank Mr. Gil Bindewald and Dr. David Ortiz from the U.S. Department of Energy Office of Electricity Delivery and Energy Reliability for their continuing support, help, and guidance. The project team appreciates technical support from Siemens Power Technologies staff Hugo Bashualdo, Dinemayer Silva, Jayapalan Senthil, James Feltes, Joseph Smith, and Krishnat Patil. We would also like to thank Siemens subcontractor Eli Pajuelo for his help in protection modeling in PSS/E.

REFERENCES

[1] U.S.–Canada Power System Outage Task Force. *Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations*. April 2004. Accessed November 1, 2015 at http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/BlackoutFinal-Web.pdf.

[2] A. Gopalakrishnan, SG Aquiles-Pérez, DM MacGregor, DB Coleman, PF McGuire, KW Jones, J Senthil, JW Feltes, G Pietrow, and A Bose. "Simulating the Smart Electric Power Grid of the 21st Century – Bridging the Gap between Protection and Planning." *Georgia Tech Protective Relaying Conference 2014*, Atlanta, Georgia. Accessed November, 1, 2015.http://quanta-technology.com/sites/default/files/doc-files/Simulating%20the%20Smart%20Grid%20paper.pdf

[3] *Generic Distance Relay Model for the Western Electricity Coordinating Council*. Published January 2014. Accessed November 1, 2015 at https://www.wecc.biz/Reliability/Distance-Relay-Model-Spec-2013-10-04.pdf.

[4] *Generator Frequency and Voltage Protective Relay Settings*. NERC Standard PRC-024-1. North American Electric Reliability Corporation, Atlanta, Georgia, 2014.

[5] Schweitzer Engineering Laboratories, Inc., "SEL-700G Family of Generator and Intertie Protection Relays," Pullman, Washington, 2015.

[6] *IEEE Standard for AC High-Voltage Circuit Breakers Rated on a Symmetrical Current Basis - Preferred Ratings and Related Required Capabilities for Voltages Above 1000 V*, IEEE Std. C37.06-2009, Nov. 2009.

[7] *System Performance Following Loss of Two or More BES Elements.* NERC standard TPL-003-0b. North American Electric Reliability Corporation, Atlanta, Georgia. Accessed November 1, 2015, at *http://www.nerc.com/files/TPL-003-0b.pdf.*

[8] W. C. New, "Load Shedding, Load Restoration and Generator Protection Using Solid-state and Electromechanical Underfrequency Relays." General Electric Company, GET-6449. (Undated).

[9] *Automatic Underfrequency Load Shedding*. NERC Standard PRC-006-NPCC-1, North American Electric Reliability Corporation, Atlanta, Georgia,

[10] *Under-Voltage Load Shedding Program Performance*. NERC Standard PRC-022-1, North American Electric Reliability Corporation, Atlanta, Georgia

Pacific Northwest
NATIONAL LABORATORY

U.S. DEPARTMENT OF
ENERGY

Proudly Operated by **Battelle** *Since 1965*

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov