PNNL-29105

Dynamic Contingency Analysis Tool 2.0 User Manual with Test System Examples

September 2019

B Vyakaranam
N Samaan
X Li
R Huang
Y Chen
M Vallem
T Nguyen
A Tbaileh
M Elizondo
X Fan
S Davis

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

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

PNNL-29105

Dynamic Contingency Analysis Tool 2.0 User Manual with Test System Examples

37T37T

September 2019

B Vyakaranam
N Samaan
X Li
R Huang
Y Chen
M Vallem
T Nguyen
A Tbaileh
M Elizondo
X Fan
S Davis

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

Pacific Northwest National Laboratory
Richland, Washington 99354

Summary

This document introduces Version 2 of the Dynamic Contingency Analysis Tool (DCAT) software package developed by Pacific Northwest National Laboratory and serves as a guide for using this package to conduct cascading failure simulations for the electric power system.

The DCAT is an open-platform and publicly available software; it is intended to help develop applications that aim to improve the capabilities of power system planning engineers to assess the impact and likelihood of extreme contingencies and potential cascading events across their systems and interconnections. Outputs from the DCAT will support finding mitigating actions to reduce the risk of cascading outages in technically sound and effective ways.

This manual provides detailed instructions on how to use this tool on multicore Windows workstations or servers. In the public version of DCAT, several improvements have been made over the previous version:[1] (1) it was modularized; (2) a graphical user interface (GUI) was developed; (3) a new version of the MPjobs parallel processing module was incorporated; and (4) a batch-processing function was added. The features of the DCAT GUI are described, including some examples of how to use the GUI to perform simulations. A full explanation of the DCAT methodology is available in the Phase 1 report .[1]

[1] Samaan NA, JE Dagle, YV Makarov, R Diao, MR Vallem, TB Nguyen, LE Miller, BG Vyakaranam, S Wang, FK Tuffner, and MA Pai. 2015. “Dynamic Contingency Analysis Tool – Phase 1.” PNNL-24843, Pacific Northwest National Laboratory, Richland, Washington. Available at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-4843.pdf.

Acronyms and Abbreviations

1D	one-dimensional
AC	alternating current
ACCC	AC contingency calculation
CPU	central processing unit
DCAT	Dynamic Contingency Analysis Tool
GUI	graphical user interface
IDE	integrated development environment
MPjobs	Python module for running PSS@E in parallel
OPF	Optimal Power Flow
PNNL	Pacific Northwest National Laboratory
PSS/E	Siemens PTI PSS/E power flow software
RAS	remedial action scheme
SPS	special protection system

Contents

Figures

Tables

1.0 Introduction

1.1 General

The bulk electric power grid is subject to vulnerabilities from component outages, which in certain combinations (extreme events) might lead to cascading outages. Cascading is a sequential disconnection of power system elements such as generators, transmission lines, and loads, potentially leading to a partial or complete blackout that leaves thousands of electricity consumers without electric power. These large blackouts have extensive effects on citizens, businesses, the economy, and the government. While such blackouts are rare, they pose a substantial risk to the security and economic health of the country. Much is known about avoiding the first few failures near the beginning of a cascade, but there is a lack of established methods for directly analyzing the risks and consequences of the longer chains of component outages. Analyzing the risks of cascading failures and devising ways to prevent them is an evolving field of study.

Pacific Northwest National Laboratory (PNNL) has developed the Dynamic Contingency Analysis Tool (DCAT) (Samaan et al. 2015) as an open-platform and publicly available methodology to improve the capabilities of power planning engineers to assess the magnitude and likelihood of extreme contingencies and potential cascading events across their systems and interconnections. DCAT is an industry-grade tool (i.e., tested and benchmarked on real-world systems) used for studying the process of cascading outages. It combines steady-state and transient simulations, manual operator actions, and the effects of the protection system, starting from an initiating event (Samaan et al. 2015). Overall, the ultimate goal of the DCAT is to bridge multiple gaps in cascading-outage analysis in a single, unique, prototype tool that can automatically simulate and analyze cascading sequences in real systems using multiprocessor computers.

The main framework of the DCAT procedure was built in Python; Figure 1.1 shows the main steps of the DCAT simulation, which will be discussed in detail in Section 6. The main procedure includes the following steps:

(1) Model preparation, which includes integrating the power flow, dynamic, and protection system models;

(2) The initial system-aggravation and event-screening module selects initiating events that trigger cascading failures;

(3) A list of contingencies is prepared;

(4) An adaptive simulation time module is implemented to run the dynamic simulation long enough to capture the response of the system. DCAT processes simulation results and logs the sequence of cascading events.

(5) Post-dynamic analysis is performed with automatic and manual corrective actions, with special protection systems (SPSs)/remedial action schemes (RASs). DCAT implements automatic control actions that eliminate voltage and flow violations using the corrective actions function of Siemens PTI PSS®E power flow software (PSS/E). Optimal power flow (OPF) and corrective actions are applied to the solved post-dynamic-simulation power flow case to mitigate the voltage and line overload violations.

(6) Overloaded lines are checked and ranked after the OPF and corrective actions. If there are no overloaded lines, the DCAT procedure will stop; otherwise, the DCAT procedure will start a new dynamic simulation by tripping the overloaded line of highest rank, and the procedure will go back to

Step (4). The developed methodology is tested using the SAVNW and Polish system power flow cases provided with PSS/E.

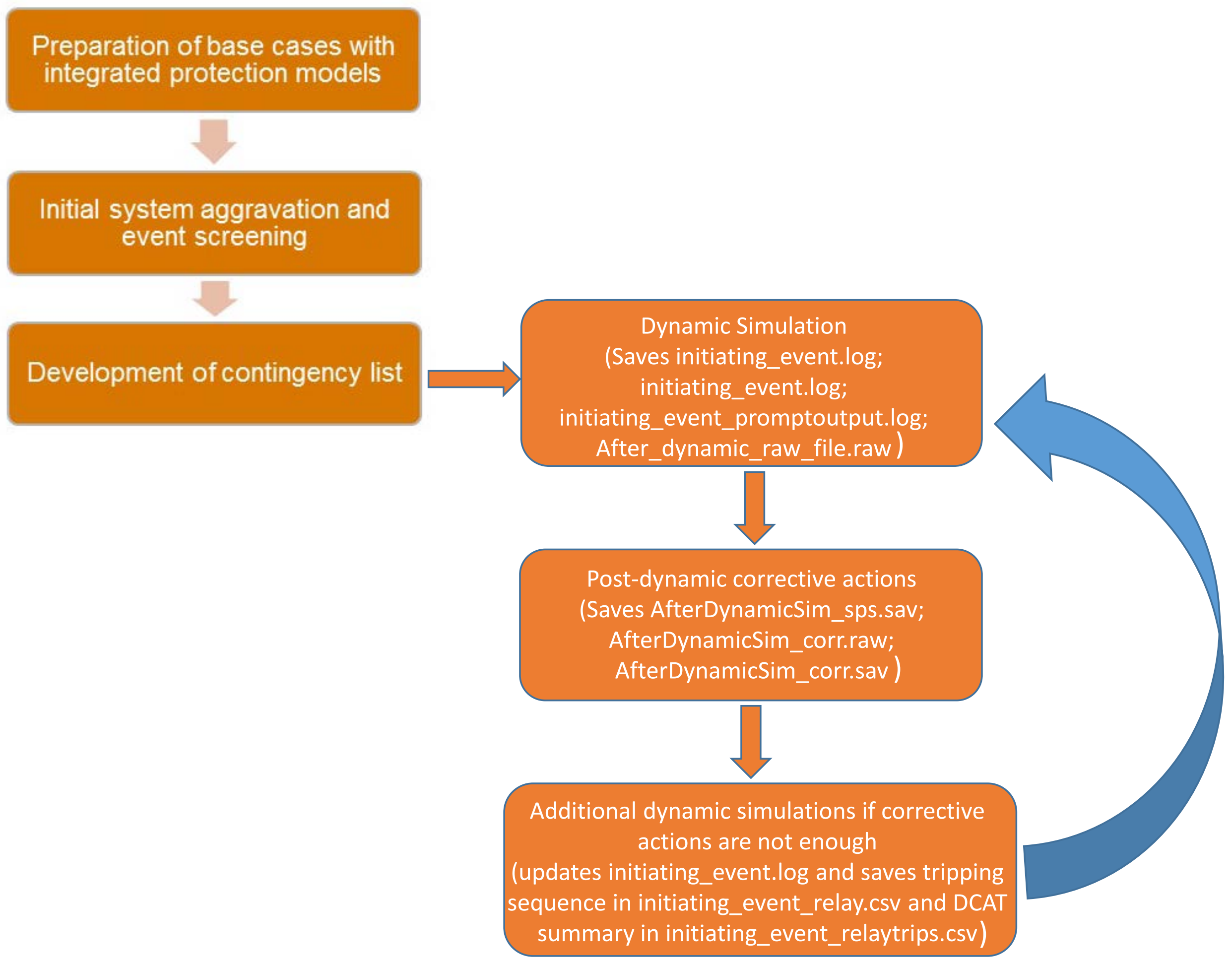

Preparation of base cases with integrated protection models
Initial system aggravation and event screening
Development of contingency list
Dynamic Simulation (Saves initiating_event.log; initiating_event.log; initiating_event_promptoutput.log; After_dynamic_raw_file.raw)
Post-dynamic corrective actions (Saves AfterDynamicSim_sps.sav; AfterDynamicSim_corr.raw; AfterDynamicSim_corr.sav)
Additional dynamic simulations if corrective actions are not enough (updates initiating_event.log and saves tripping sequence in initiating_event_relay.csv and DCAT summary in initiating_event_relaytrips.csv)

Figure 1.1. Main steps of DCAT simulation

This manual provides detailed instructions on how to use DCAT to perform dynamic contingency analysis on multicore Windows workstations or servers. This version of DCAT incorporates several improvements over the previous version of DCAT (Samaan et al. 2015): (1) it has been modularized; (2) a GUI has been developed; (3) a new version of MPjobs is used; and (4) a batch-processing function has been added. The features of the DCAT GUI are described, including some examples of how to use the GUI to perform simulations. A full explanation of the DCAT methodology is available in the Phase 1 report (Samaan et al. 2015).

1.2 Outline

This manual is organized as follows:

Section 2 contains system requirements for running the DCAT package.

Section 3 provides an introduction to the DCAT package. It then describes various source code files and folders in the DCAT package.

Section 4 presents the various steps for running the DCAT package using the GUI.

Section 5 presents DCAT results from several examples using the PSS/E SAVNW case or the Polish system. 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. The examples also show how planning engineers can use DCAT for cascading-outage analysis and how the results are reported.

Section 6 introduces details of various Python modules in the DCAT package. It then describes how DCAT users can use the modules independently.

Section 7 provides generic information on data preparation, configuration settings, and scripting for various DCAT runs.

2.0 System Requirements

The system requirements for running the DCAT package are as follows:

A. Windows 7 or later (64 bit)

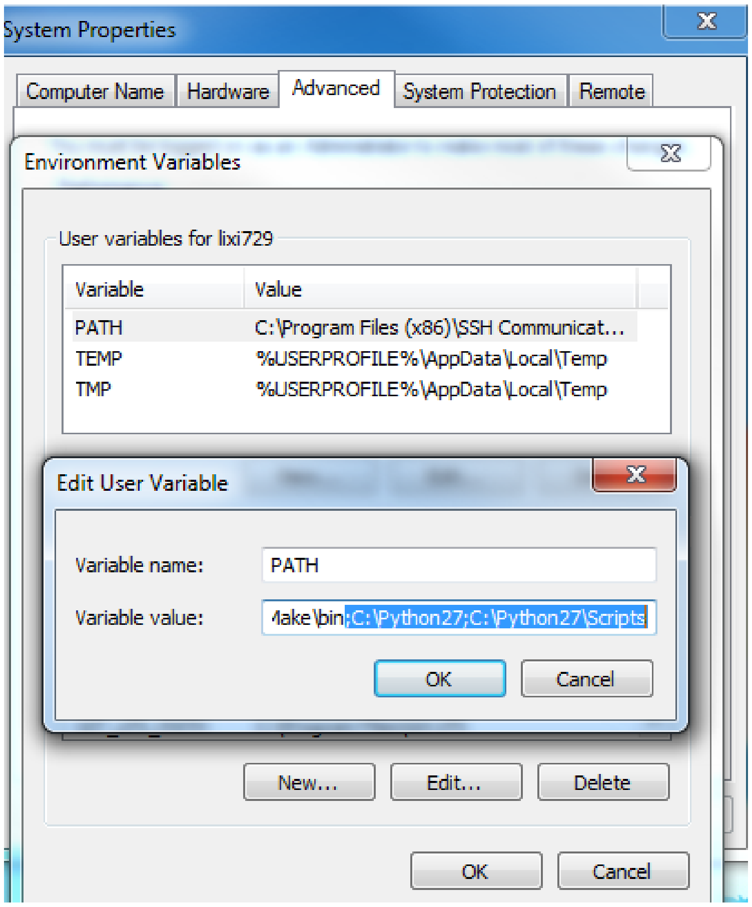
B. Python 2.7 (32 bit)

a. How is Python 2.7 set up?

When PSS/E is installed, it will install Python on the local computer. However, this automatically installed Python version might be older than 2.7.9. It is suggested that the user installs a Python version 2.7.9 (or higher) with the "pip" package.

Download the latest version for Windows of Python 2.7.XX from https://www.python.org/downloads/. Python will be installed by default to the path "C:\Python27".

Append the installation path (e.g., ";C:\Python27;C:\Python27\Scripts" —semicolons are used to separate different entries) to the "PATH" variable in System variables (In Windows 7, open Computer > Properties >Advanced System Settings > Environment Variables), as shown in Figure 2.1.



Variable	Value
PATH	C:\Program Files (x86)\SSH Communicat...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

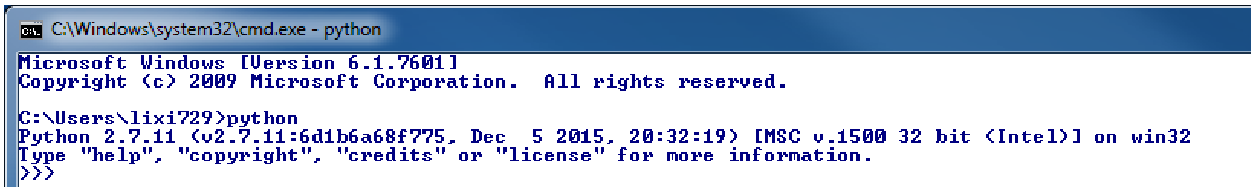
Variable name: PATH

Variable value: Make\bin;C:\Python27;C:\Python27\Scripts

Figure 2.1. Installing the latest version of Python

For more details, please follow the online tutorial, “Using Python on Windows” (https://docs.python.org/2.7/using/windows.html).

Now the user is able to run the “*.py” file directly by double-clicking it or from Windows Command Prompt, “cmd,” as shown in Figure 2.2.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\lixi729>python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:32:19) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 2.2. Running Python with a Windows command prompt

To execute a Python script, type the following command in “cmd:”

```
python pathname\filename.py
```

b. How are required packages installed?

“**NumPy**” is a fundamental package for scientific computing with Python. It is required to execute DCAT. To install a package/lib to the local system, the recommended method is to use the Python installation tool, “**pip**,” which is already installed with Python 2.7 (version ≥ 2.7.9). To install the latest version of NumPy, type the follow command in cmd:

```
pip install numpy
```

For more details, please follow the online tutorial, “Installing Packages”: (https://packaging.python.org/tutorials/installing-packages/).

“**DYNTOOLS”** (optional) is used to export output channels from a PSS/E OUT file. DYNTOOLS is an internal module for PSS/E that can be loaded without prior installation, similar to PSSPY.

“**MATPLOTLIB**” (optional) is a Python plotting library that produces publication-quality figures in a variety of formats and interactive environments across platforms. To install the latest version of MATPLOTLIB, type the following command in cmd:

```
pip install matplotlib
```

More details can be found in Appendix A.

C. Siemens PSS/E 33.7/33.12/34.5

An important feature is that DCAT Version 2.0 works with both PSS/E versions 33 and 34. The examples described in Section 5.0 were verified with PSS/E versions 33.7, 33.12, and 34.5. DCAT will switch between PSS/E 33 and 34 according to the user’s choice.

After PSS/E 33 is installed in the default path (e.g., “C:\Program Files (x86)\PTI\PSSE33\PSSBIN”), in a Python script, the psspy module is imported as follows:

```
PSSE_PATH = 'C:\\Program Files (x86)\\PTI\\PSSE33\\PSSBIN'
sys.path.append(PSSE_PATH)
```

```
os.environ['PATH'] += ';' + PSSE_PATH
import psspy
```

After PSS/E 34 is installed in the default path (e.g., "C:\Program Files (x86)\PTI\PSSE34\PSSBIN"), in a Python script, the psspy module is imported as follows:

```
import psse34
import psspy
```

D. Eclipse (optional)

Eclipse or another integrated development environment (IDE) is highly recommended. The recommended Eclipse packages are

- Eclipse IDE for Eclipse Committers
- Eclipse IDE for Java EE Developers.

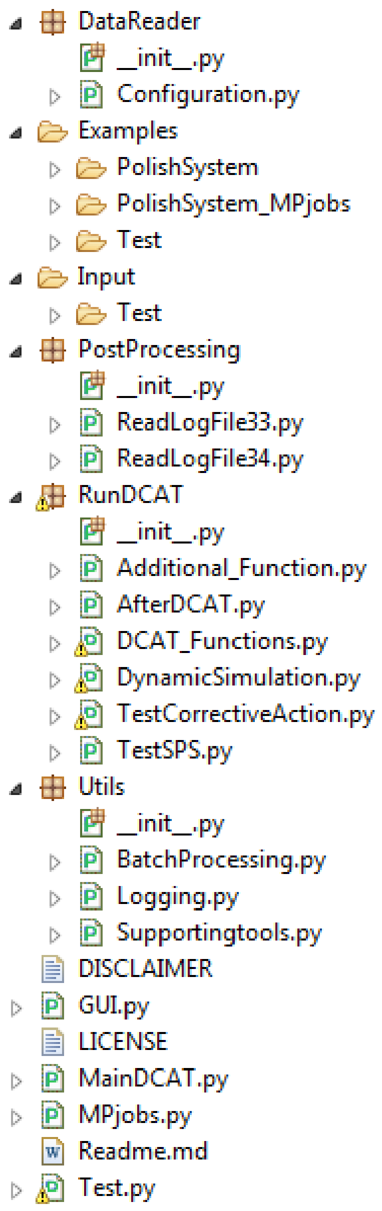
Follow the download instructions on this version web page: https://www.eclipse.org/downloads/packages/

Link Eclipse with the PyDev plug-in to run Python script as a PyDev project. PyDev requires Java 7 in order to run. More details can be found in Appendix A.

E. PyCharm IDE (optional): The community and educational versions of PyCharm are open-source and they are free. Here is the link to install PyCharm: https://www.jetbrains.com/pycharm/

3.0 Introduction to the DCAT Package

This section introduces the DCAT package. The full DCAT package, developed by PNNL, consists of source code files (four module packages [DataReader, PostProcessing, RunDCAT, Utils], the Main module, the MPjobs module, and the graphical user interface [GUI] module), an input folder, and a folder for example cases. Figure 3.1 shows the DCAT folder structure and source code files along with the individual modules. These modules can be used independently by the user. Table 3.1 summarizes DCAT modules and related folders for inputs and outputs. Every module package contains several module files and each module file (*.py) contains several functions. A function declares a computational step in a DCAT simulation, which provides modularity for DCAT applications. Details on the package breakdown are provided in the next portion of the document.



```
DataReader
    __init__.py
    Configuration.py
Examples
    PolishSystem
    PolishSystem_MPjobs
    Test
Input
    Test
PostProcessing
    __init__.py
    ReadLogFile33.py
    ReadLogFile34.py
RunDCAT
    __init__.py
    Additional_Function.py
    AfterDCAT.py
    DCAT_Functions.py
    DynamicSimulation.py
    TestCorrectiveAction.py
    TestSPS.py
Utils
    __init__.py
    BatchProcessing.py
    Logging.py
    Supportingtools.py
DISCLAIMER
GUI.py
LICENSE
MainDCAT.py
MPjobs.py
Readme.md
Test.py
```

Figure 3.1. DCAT (2019 version) after modularization

Table 3.1. Organized DCAT modules and related folders for inputs and outputs

DCAT Architecture		
Category	Layer I	Layer II
	DataReader	Configuration.py
		DCAT_Functions.py
		DynamicSimulation.py
	RunDCAT	TestSPS.py
		TestCorrectiveAction.py
		AfterDCAT.py
		ReadLogFile33.py
Source Code	PostProcessing	ReadLogFile34.py
		BatchProcessing.py
	Utils	Logging.py
		Supportingtools.py
	GUI.py	
	MPjobs.py	
	MainDCAT.py	
	Input	Test
Inputs and Examples	Examples	Test
		PolishSystem
		PolishSystem_MPjobs
🟨	Module Package	
🟩	Module	
🟦	Folder of Common Input Files	

The source code files were categorized into several module packages, besides the main function and the GUI module:

1. “DataReader”: stores the modules related to the input/output (I/O) interface, such as constructing the class of configuration settings as the input parameters of “MainDCAT.”
2. “RunDCAT”: stores the core computational modules of DCAT.
3. “PostProcessing”: stores the modules related to post-processing after the major results files are output by PSS/E, such as extracting useful information (relay tripping) from the log files generated and generating plots for channels of interest.
4. “Utils”: stores the supporting tools (useful functions) and utility modules, such as batch processing.
5. “MPjobs.py”: the MPjobs (parallel processing) module.
6. “GUI.py”: the GUI module.

7. “MainDCAT.py”: also known as “MainDCAT”, the main module of the DCAT tool.
8. “Input”: The input files needed for the example cases are stored in the folder “Input”. Five types of files are required as the common (static) inputs for example cases: “inl” file (inertial power flow), “sub” file and “mon” file (these two will be adopted to conduct AC contingency calculation [ACCC]), and the “sav” file and “snp” file (the network files and dynamic files to be used in the DCAT process, respectively). The paths and names of these files should be included in the configuration file.
9. The “Example” folder is divided into three subfolders:
    a. “Test”: Five examples for the SAVNW system, as described in Section 5.0.
    b. “PolishSystem”: Four examples for the Polish system, as described in Section 5.0.
    c. “PolishSystem_MPjobs”: An example of one-dimensional (1D) MPjobs (Section 6.4) with six cases of the Polish system. In this folder, configuration files for PSS/E 33 and PSS/E 34 are included separately. The only difference between these two .ini files is that the parameter “PSSEPath” is different (see Section 7.2).

3.1 Model Preparation

The model preparation is a very important element of the DCAT methodology. The user can provide multiple base planning cases corresponding, for example, to different seasons, different levels of wind and solar penetration, different load levels, variants of possible system reinforcements, etc., reflecting a variety of possible initial system conditions. The base planning cases include both power flow and dynamic system models. Protection system models are added to the base cases. This section describes how to prepare base cases and dynamic snapshot data files with protection models integrated.

3.1.1 Convert and Modify the Base Case

The Python module “Make_DYR_withPNNLrelays_xx.py” (for test system “SAVNW”) should be called to convert the power flow model and to modify any topology or network parameters. All the preprocessing scripts along with all the supporting files are saved in the “Preprocessing” folder. 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 revised the base case, including modifying the tap ratio, the rating of some transformers, and the Rate C of the branches. Users can disable those changes by simply “commenting” them out.
4. Finally, the converted case will be saved.

The preprocessing folder has three modules: “Make_DYR_withPNNLrelays_01.py,” “Make_DYR_withPNNLrelays_02.py,” and “Make_DYR_withPNNLrelays_35.py.” The user has to choose and run the correct “Make_DYR_withPNNLrelays _xx.py” script as shown in Table 3.2.

Table 3.2. Options for Make_DYR_withPNNLrelays _xx.py

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

3.1.2 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 is one of the application program interfaces 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 users can modify the name of the file.

4.0 How to Run DCAT Examples

This section presents the various steps for running the DCAT package on the examples provided. After all required packages are installed, there are seven main steps to run this DCAT package:

Step 1: Verify files

This step is to make sure that all the necessary files are ready and located in the correct folders. Refer to Section 3.0 for the specific folder structure and required files. For the Test system, the detailed folder structure of the DCAT package is shown in Figure 4.1. The "Examples" folder has five examples on the SAVNW system ("Test") and four examples on the Polish system ("PolishSystem"). Each example folder includes files needed for running the DCAT process. It has "snp," ".sav," ".py," ".ini," and ".idv" files.

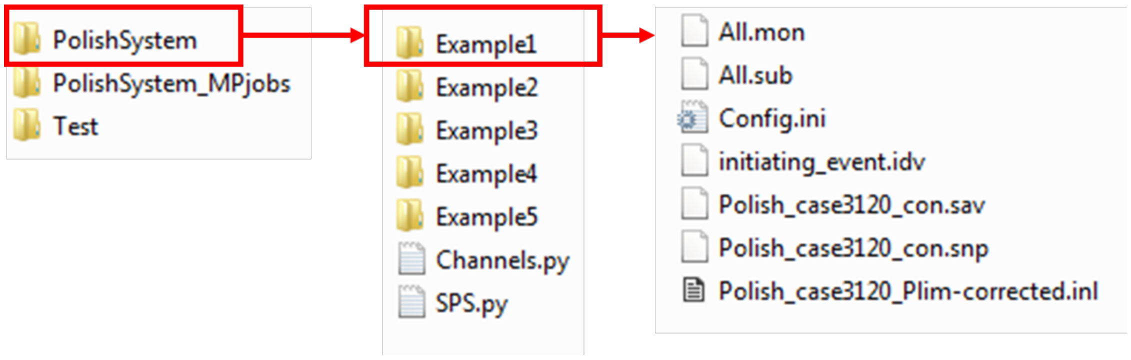
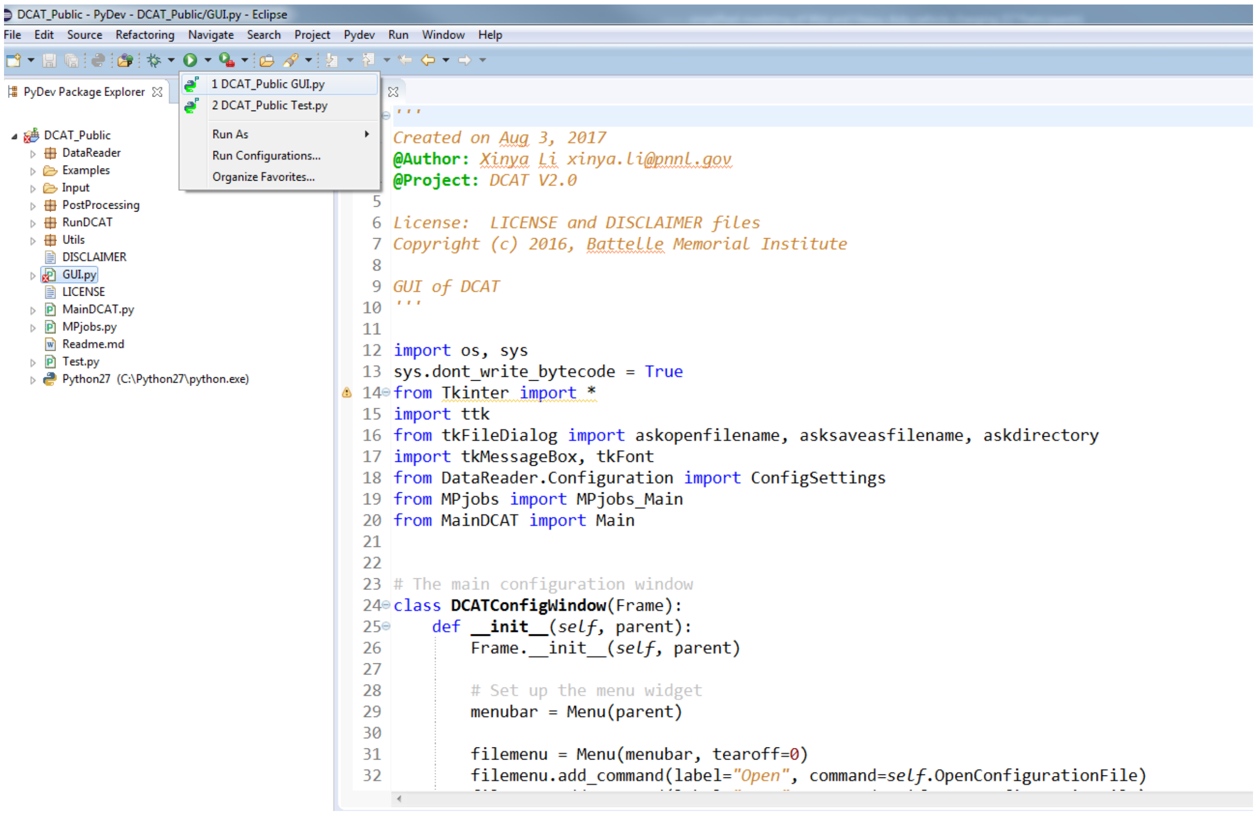


Figure 4.1. DCAT folder structure

Step 2: Open and run GUI.py

Users can run the DCAT script "GUI.py" either by double-clicking this file in the main folder, or by running the script using any Python IDE. In this document, PyDev for Eclipse was used to run "GUI.py" and is shown in Figure 4.2.

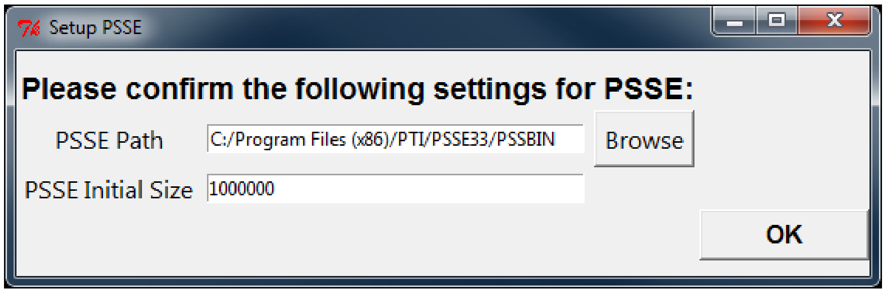


```
'''
Created on Aug 3, 2017
@Author: Xinya Li xinya.li@pnnl.gov
@Project: DCAT V2.0

License:  LICENSE and DISCLAIMER files
Copyright (c) 2016, Battelle Memorial Institute

GUI of DCAT
'''

import os, sys
sys.dont_write_bytecode = True
from Tkinter import *
import ttk
from tkFileDialog import askopenfilename, asksaveasfilename, askdirectory
import tkMessageBox, tkFont
from DataReader.Configuration import ConfigSettings
from MPjobs import MPjobs_Main
from MainDCAT import Main


# The main configuration window
class DCATConfigWindow(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)

        # Set up the menu widget
        menubar = Menu(parent)

        filemenu = Menu(menubar, tearoff=0)
        filemenu.add_command(label="Open", command=self.OpenConfigurationFile)
```

Figure 4.2. Running GUI.py using Python IDE

Step 3: Set up PSS/E

When the GUI.py is run, a PSS/E set-up window will pop up, as shown in Figure 4.3. The user should confirm the PSS/E Path and Initial Size.



Please confirm the following settings for PSSE:

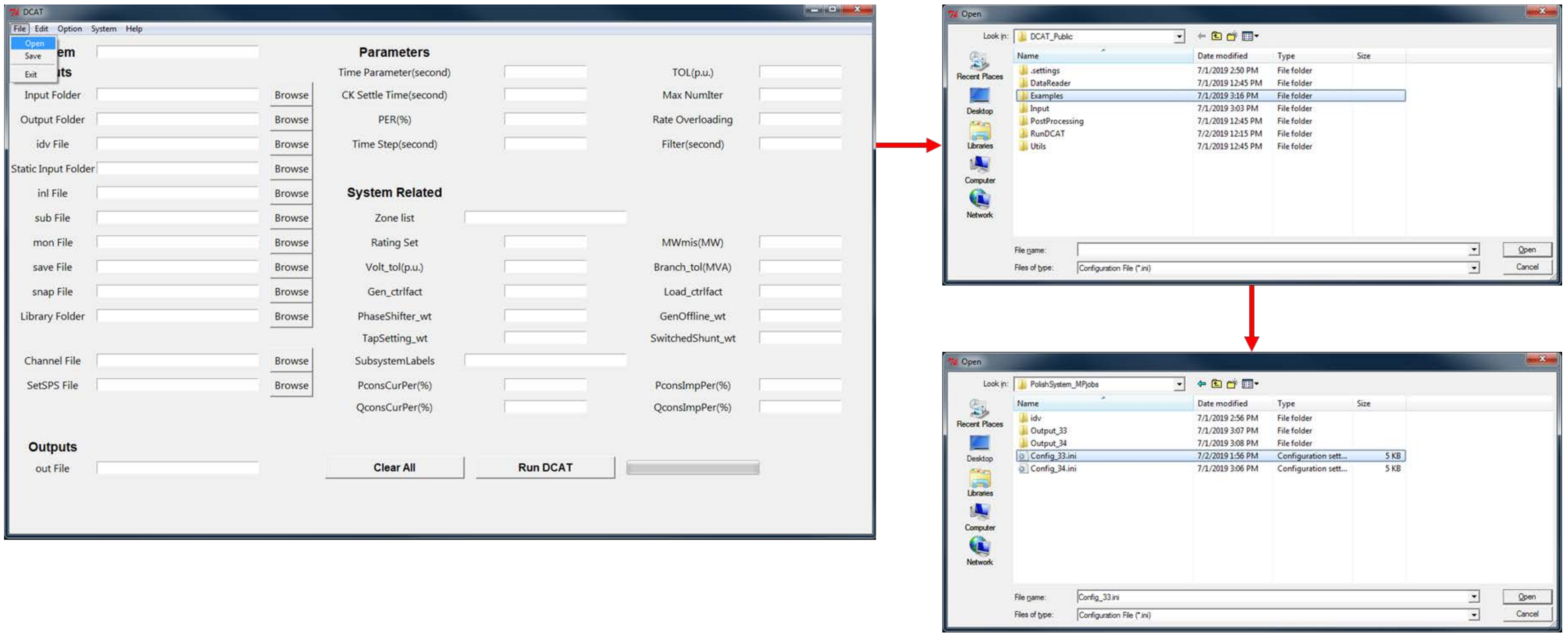
PSSE Path: C:/Program Files (x86)/PTI/PSSE33/PSSBIN

PSSE Initial Size: 1000000

Figure 4.3. PSS/E settings

Step 4: Import configuration settings

The user can open an existing configuration file (*.ini) using the command (“File” → “Open”), as shown in Figure 4.4.



Look in: DCAT_Public

Name	Date modified	Type	Size
.settings	7/1/2019 2:50 PM	File folder	
DataReader	7/1/2019 12:45 PM	File folder	
Examples	7/1/2019 3:16 PM	File folder	
Input	7/1/2019 3:03 PM	File folder	
PostProcessing	7/1/2019 12:45 PM	File folder	
RunDCAT	7/2/2019 12:15 PM	File folder	
Utils	7/1/2019 12:45 PM	File folder	

File name:
Files of type: Configuration File (*.ini)

Look in: PolishSystem_MPjobs

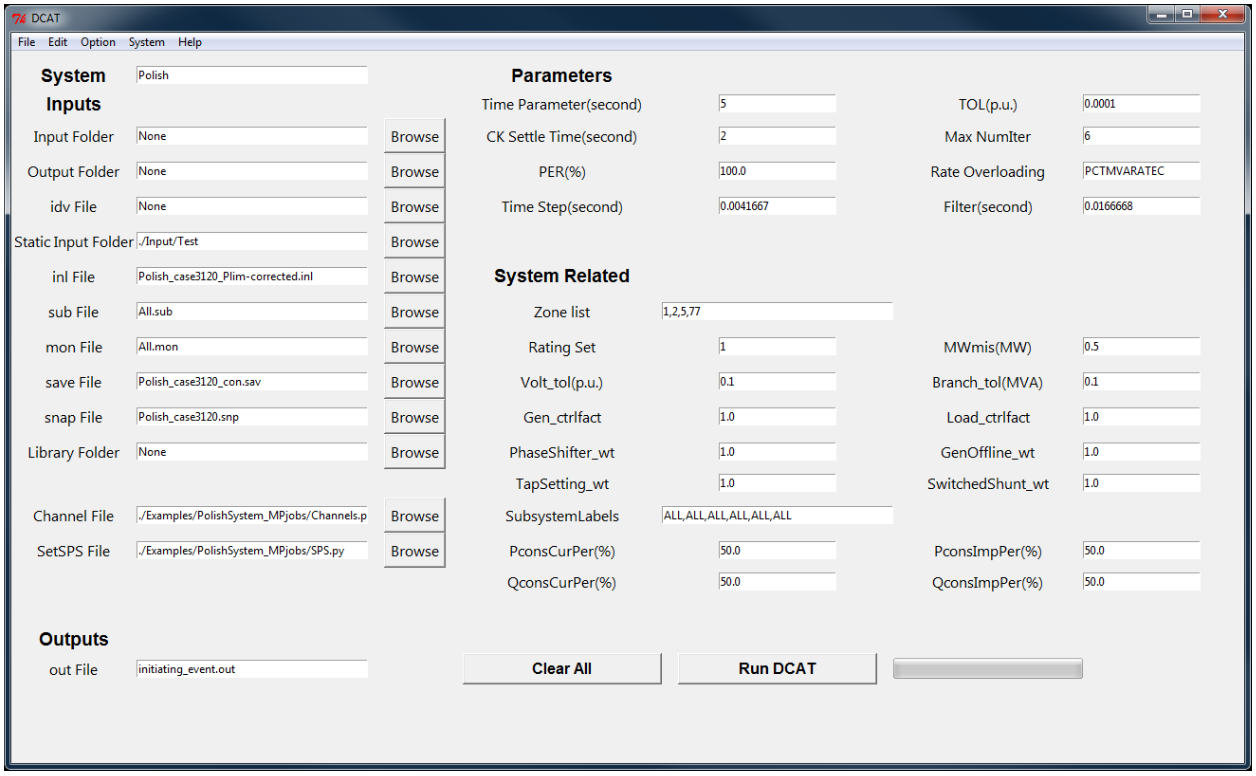
Name	Date modified	Type	Size
idv	7/1/2019 2:56 PM	File folder	
Output_33	7/1/2019 3:07 PM	File folder	
Output_34	7/1/2019 3:08 PM	File folder	
Config_33.ini	7/2/2019 1:56 PM	Configuration sett...	5 KB
Config_34.ini	7/1/2019 3:06 PM	Configuration sett...	5 KB

File name: Config_33.ini
Files of type: Configuration File (*.ini)

Figure 4.4. Importing configuration settings

Step 5: Modify the configuration parameters (optional)

The control interface between the user and the DCAT package is a configuration file. Each simulation requires a unique configuration file defined by the user. Once the user selects the configuration file, a DCAT window will pop up, as shown in Figure 4.5. The user can modify the inputs of configuration or clear all the inputs using the button “Clear All.” The inputs shown in the GUI window can be saved into a new “ini” file or replace an existing “ini” file using the command “File>Save” from the menu. “save file” and “snap file” are the two required input files for DCAT. For MPjobs mode, an additional window will pop up as shown in Figure 4.5. The user is able to modify the related parameters controlling MPjobs mode, such as how many cores are to be used.



System Polish

Inputs

Input Folder	None
Output Folder	None
idv File	None
Static Input Folder	./Input/Test
inl File	Polish_case3120_Plim-corrected.inl
sub File	All.sub
mon File	All.mon
save File	Polish_case3120_con.sav
snap File	Polish_case3120.snp
Library Folder	None
Channel File	./Examples/PolishSystem_MPjobs/Channels.p
SetSPS File	./Examples/PolishSystem_MPjobs/SPS.py

Parameters

Time Parameter(second)	5	TOL(p.u.)	0.0001
CK Settle Time(second)	2	Max NumIter	6
PER(%)	100.0	Rate Overloading	PCTMVARATEC
Time Step(second)	0.0041667	Filter(second)	0.0166668

System Related

Zone list	1,2,5,77		
Rating Set	1	MWmis(MW)	0.5
Volt_tol(p.u.)	0.1	Branch_tol(MVA)	0.1
Gen_ctrlfact	1.0	Load_ctrlfact	1.0
PhaseShifter_wt	1.0	GenOffline_wt	1.0
TapSetting_wt	1.0	SwitchedShunt_wt	1.0
SubsystemLabels	ALL,ALL,ALL,ALL,ALL,ALL		
PconsCurPer(%)	50.0	PconsImpPer(%)	50.0
QconsCurPer(%)	50.0	QconsImpPer(%)	50.0

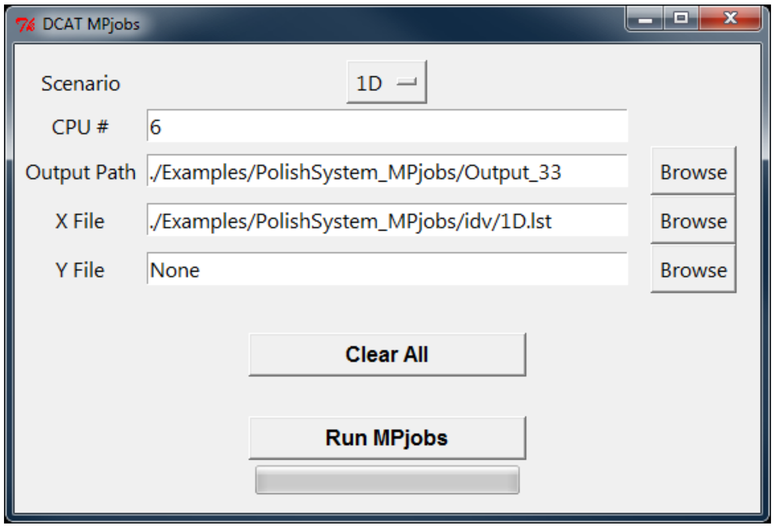
Outputs

out File initiating_event.out

Figure 4.5. Input and output parameters for a single DCAT simulation

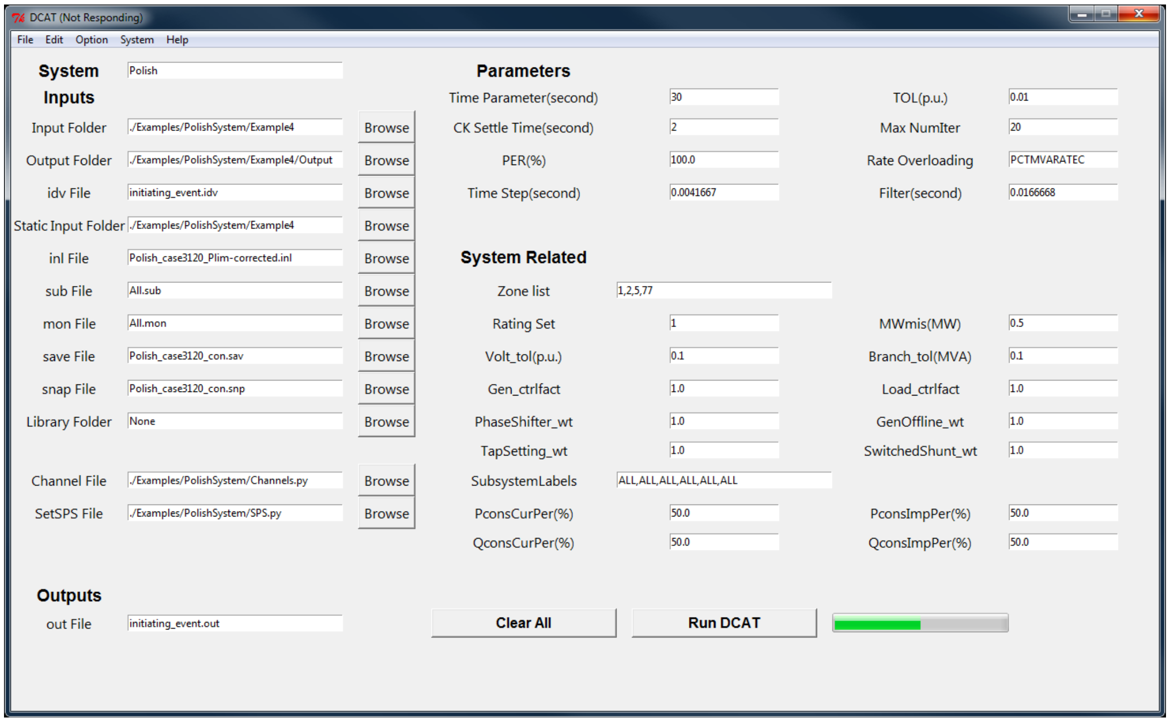
Step 6: “Run DCAT” or “Run MPjobs”

To initiate a single DCAT simulation, user should select “Run DCAT” option as shown in Figure 4.5. To initiate DCAT simulations using MPjobs, user should select “Run MPjobs” option as shown in Figure 4.6. When DCAT is running, a progress bar is displayed next to “Run DCAT” (Figure 4.7, Top), and when DCAT finished, a dialogue window will pop up (Figure 4.7, Bottom).

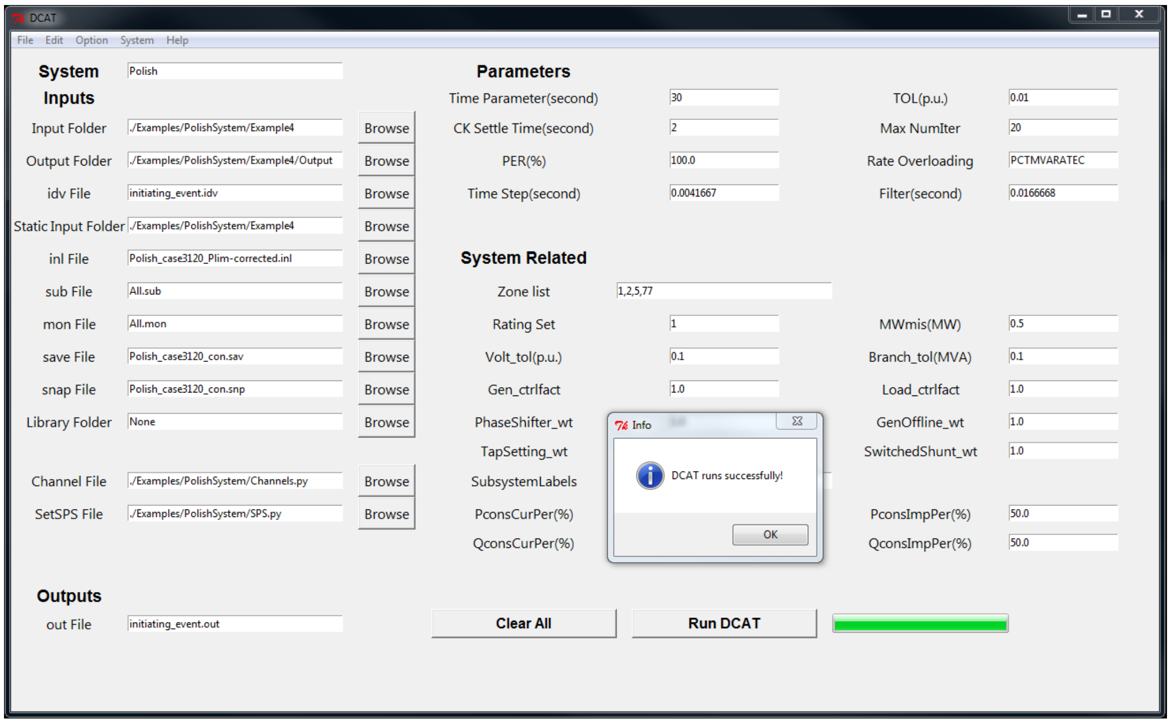


Scenario: 1D
CPU #: 6
Output Path: ./Examples/PolishSystem_MPjobs/Output_33
X File: ./Examples/PolishSystem_MPjobs/idv/1D.lst
Y File: None

Figure 4.6. Input and output parameters for running MPjobs



System	Polish	**Parameters**			
Inputs		Time Parameter(second)	30	TOL(p.u.)	0.01
Input Folder	./Examples/PolishSystem/Example4	CK Settle Time(second)	2	Max NumIter	20
Output Folder	./Examples/PolishSystem/Example4/Output	PER(%)	100.0	Rate Overloading	PCTMVARATEC
idv File	initiating_event.idv	Time Step(second)	0.0041667	Filter(second)	0.0166668
Static Input Folder	./Examples/PolishSystem/Example4				
inl File	Polish_case3120_Plim-corrected.inl	**System Related**			
sub File	All.sub	Zone list	1,2,5,77		
mon File	All.mon	Rating Set	1	MWmis(MW)	0.5
save File	Polish_case3120_con.sav	Volt_tol(p.u.)	0.1	Branch_tol(MVA)	0.1
snap File	Polish_case3120_con.snp	Gen_ctrlfact	1.0	Load_ctrlfact	1.0
Library Folder	None	PhaseShifter_wt	1.0	GenOffline_wt	1.0
		TapSetting_wt	1.0	SwitchedShunt_wt	1.0
Channel File	./Examples/PolishSystem/Channels.py	SubsystemLabels	ALL,ALL,ALL,ALL,ALL,ALL		
SetSPS File	./Examples/PolishSystem/SPS.py	PconsCurPer(%)	50.0	PconsImpPer(%)	50.0
		QconsCurPer(%)	50.0	QconsImpPer(%)	50.0
Outputs					
out File	initiating_event.out				



System	Polish	**Parameters**			
Inputs		Time Parameter(second)	30	TOL(p.u.)	0.01
Input Folder	./Examples/PolishSystem/Example4	CK Settle Time(second)	2	Max NumIter	20
Output Folder	./Examples/PolishSystem/Example4/Output	PER(%)	100.0	Rate Overloading	PCTMVARATEC
idv File	initiating_event.idv	Time Step(second)	0.0041667	Filter(second)	0.0166668
Static Input Folder	./Examples/PolishSystem/Example4				
inl File	Polish_case3120_Plim-corrected.inl	**System Related**			
sub File	All.sub	Zone list	1,2,5,77		
mon File	All.mon	Rating Set	1	MWmis(MW)	0.5
save File	Polish_case3120_con.sav	Volt_tol(p.u.)	0.1	Branch_tol(MVA)	0.1
snap File	Polish_case3120_con.snp	Gen_ctrlfact	1.0	Load_ctrlfact	1.0
Library Folder	None	PhaseShifter_wt		GenOffline_wt	1.0
		TapSetting_wt		SwitchedShunt_wt	1.0
Channel File	./Examples/PolishSystem/Channels.py	SubsystemLabels			
SetSPS File	./Examples/PolishSystem/SPS.py	PconsCurPer(%)		PconsImpPer(%)	50.0
		QconsCurPer(%)		QconsImpPer(%)	50.0
Outputs					
out File	initiating_event.out				

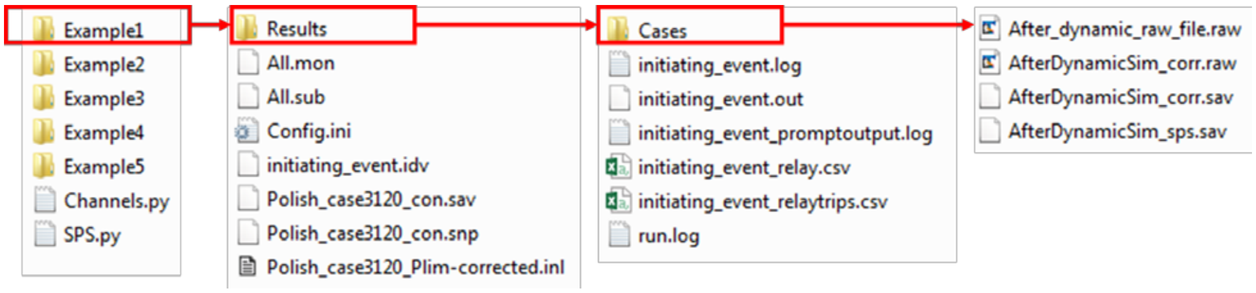
DCAT runs successfully!

Figure 4.7. Display while DCAT is running (top) and when DCAT simulation is finished (bottom)

Step 7: Collecting Output Files

The detailed folder structure of the DCAT package including output files is shown in Figure 4.8. A new folder is created under each example folder and DCAT simulation results are saved under the new folder.

Power flow cases at each stage of the dynamic contingency process will be saved under the “Case” subfolder.



Example1
Example2
Example3
Example4
Example5
Channels.py
SPS.py

Results
All.mon
All.sub
Config.ini
initiating_event.idv
Polish_case3120_con.sav
Polish_case3120_con.snp
Polish_case3120_Plim-corrected.inl

Cases
initiating_event.log
initiating_event.out
initiating_event_promptoutput.log
initiating_event_relay.csv
initiating_event_relaytrips.csv
run.log

After_dynamic_raw_file.raw
AfterDynamicSim_corr.raw
AfterDynamicSim_corr.sav
AfterDynamicSim_sps.sav

Figure 4.8. Output files

Channel output files are produced by PSS/E dynamic simulation activities. The user specifies the type of data to be placed in output channels, and the element to be monitored. To view dynamic results, a Plot Book has to be created. After a Plot Book is created, a channel output file has to be opened. Any channel output quantity can be viewed by dragging and dropping into the Plot Page. The example figures showed in Plot Page using PSS/E can be found in Chapter 5.0.

5.0 DCAT Case Studies

This section presents simulation results for DCAT with several examples using a test system (savnw.sav) that is provided with the PSS/E software package and using the Polish system. 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.

5.1 Test System 1: PSS/E Test System

Simulation tests are performed with the DCAT by considering the PSS/E test system (savnw.sav). This test system has 23 buses and six power plants. Figure 5.1 shows a one-line diagram of the test system. Five different simulation tests are performed here using the test system, and outcomes of each test are presented in the following subsections. Table 5.1 compares different examples.

Table 5.1. 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	NA	No	NA	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	NA	No	NA	None
5.1.3	PSS/E test system “savnw”	3 phase bus fault for 10 cycles	Yes (75 s) 2 relay actions	NA	No	NA	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)	NA	NA	NA	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

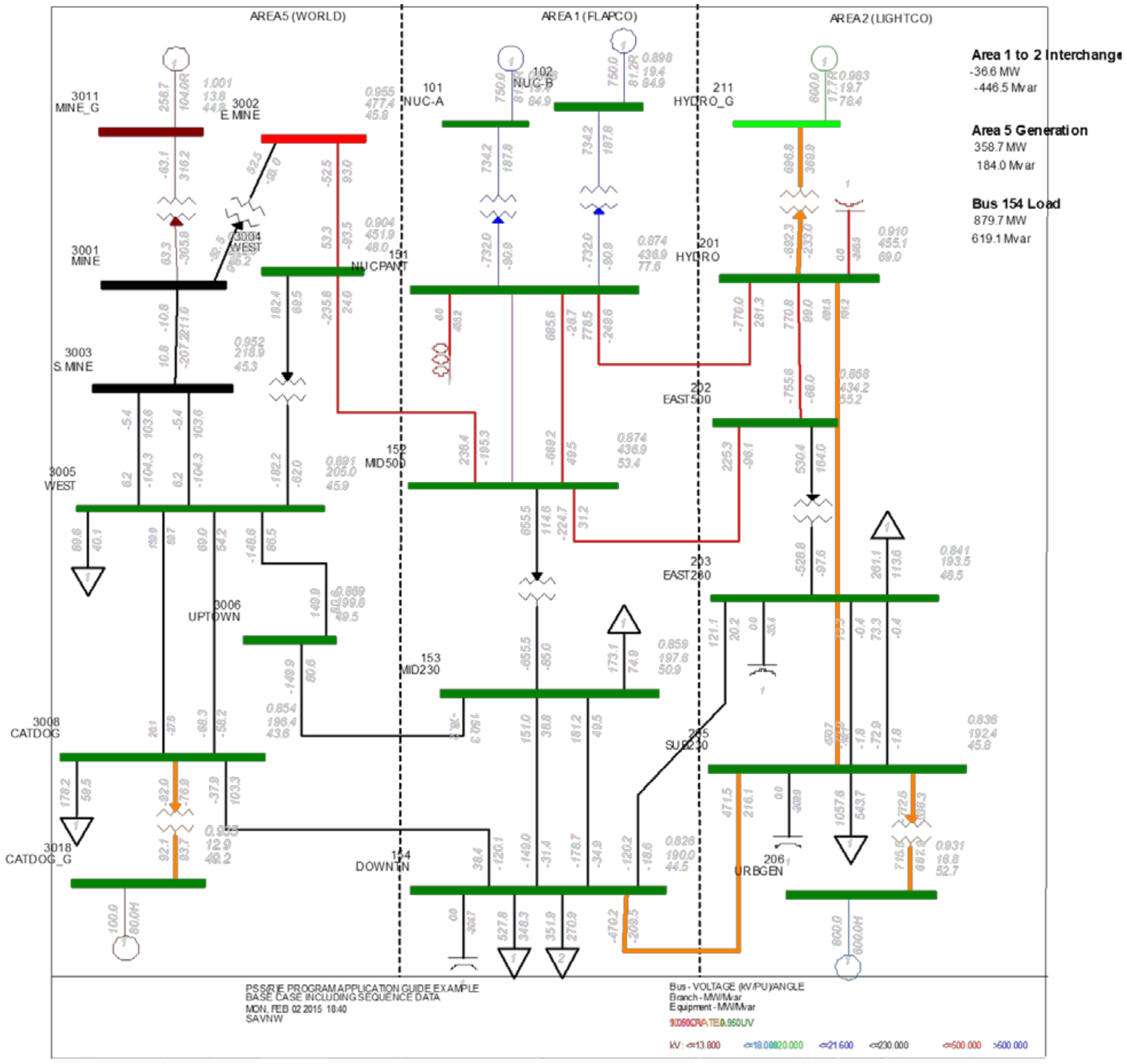


Figure 5.1. One-line diagram of the “savnw” test system

5.1.1 Example 1 in “savnw” System: 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 the 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. 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 5.2 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 extends from the near end to the fictitious bus, and the other extends from the remote end to the fictitious bus.

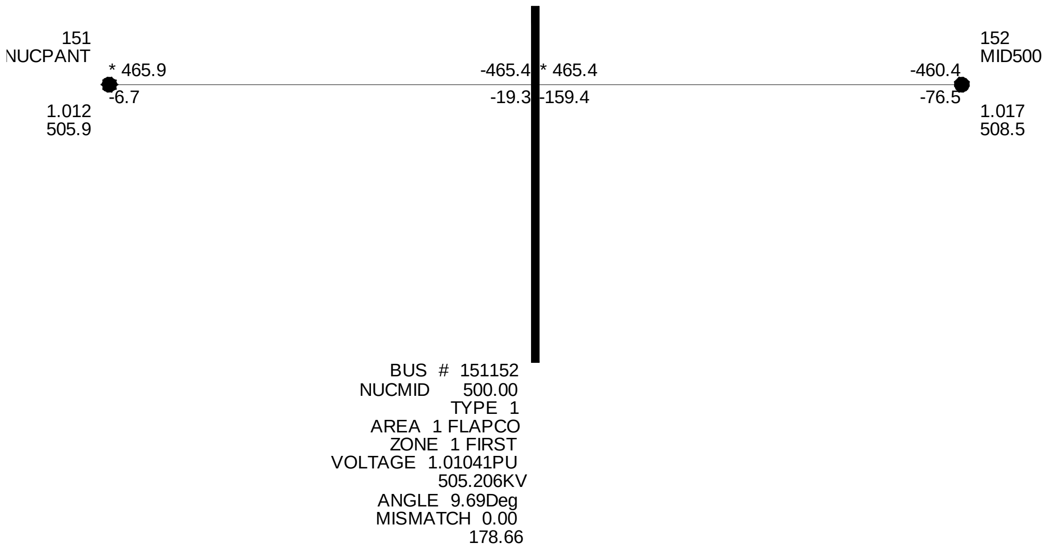


Figure 5.2. A fictitious bus between Buses 151 and 152

The bus fault is introduced at the fictitious bus (151152) at t = 5 s and the 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. The distance relay (DISTR1) at Circuit 1 from Bus 151 to Bus 151152 is activated as Zone 1 and its timer starts at t = 5 s.
2. The distance relay (DISTR1) at Circuit 1 from Bus 152 to Bus 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; at the same time, the self-trip breaker timer and the transfer trip and breaker trip timers start.
4. Circuit 1 from Bus 151 to Bus 151152 trips at t = 5.05 s and the transfer trip timer 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 Bus 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 5.3 shows that the voltage at Bus 151 collapses farther than the Bus 152 voltage. This indicates that the fault is closer to Bus 151.

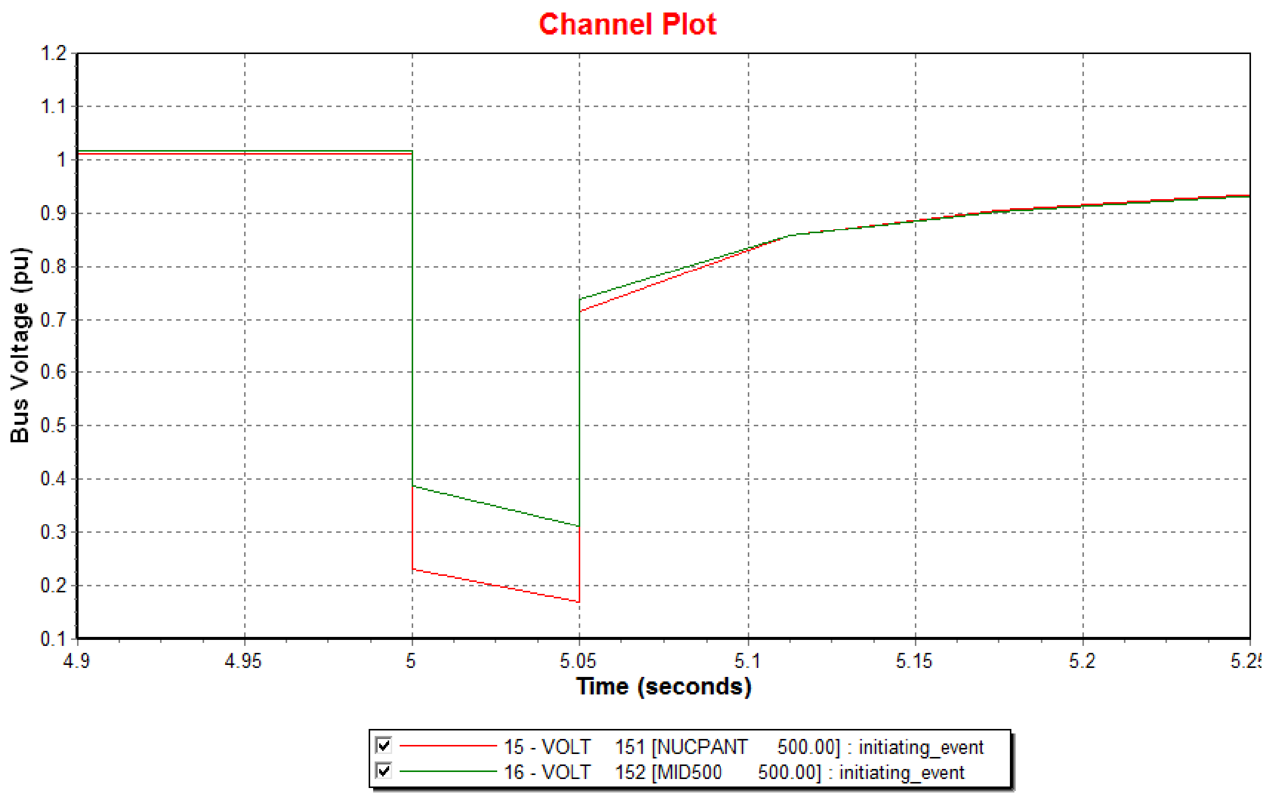


Figure 5.3. Voltage plots of the terminal buses of the faulted line for Test 1

5.1.2 Example 2 in “savnw” System: 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 end of the line nearer to the fault at Bus 151 trips on the Zone 1 setting (four 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. The distance relay (DISTR1) at Circuit 1 from Bus 151 to Bus 151152 is activated as Zone 1 and its timer starts at *t* = 5 s.
2. The distance relay (DISTR1) at Circuit 1 from Bus 152 to Bus 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; at the same time, the self-trip breaker timer and breaker timer start.
4. Circuit 1 from Bus 151 to Bus 151152 trips at *t* = 5.05 s.
5. Circuit 1 from Bus 152 to Bus 151152 trips as Zone 2 fault at *t* = 5.333 s and the channel plot (Figure 5.4) shows the two voltages start to recover after tripping both ends of the branch.

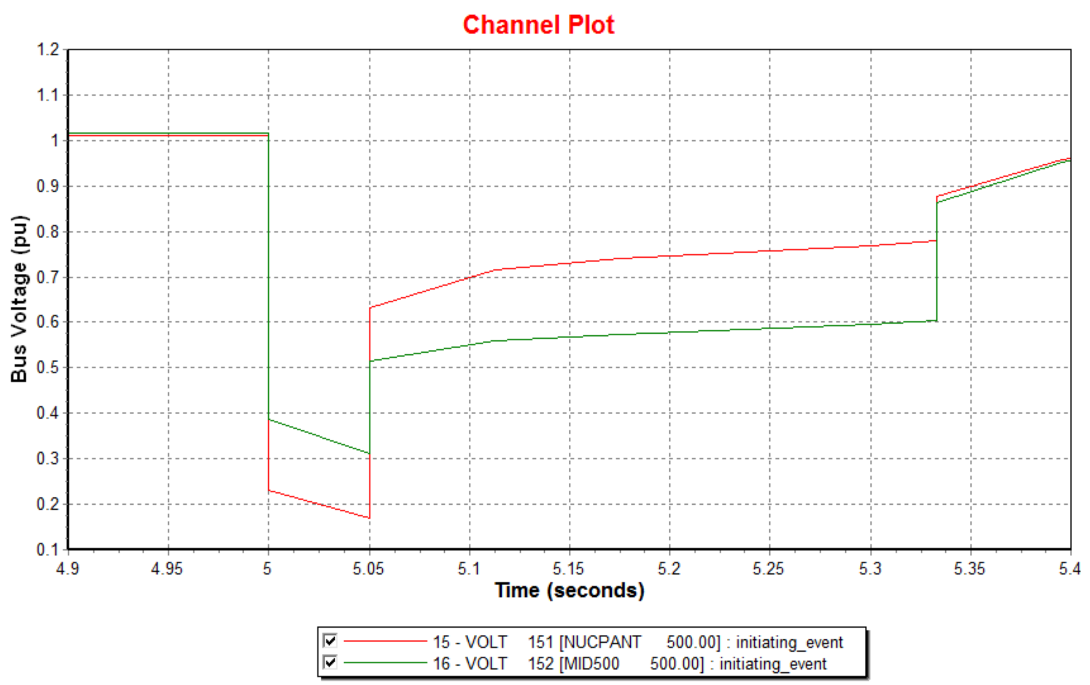


Figure 5.4. Channel plot for Test 2

5.1.3 Example 3 in “savnw” System: 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 5.2 shows a summary of this activity. 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 5.3. Simulation result plots are shown in Figure 5.5 and Figure 5.6.

Table 5.2. Relay trips summary for 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	Gen ID	GenName	GenkV
VTGTPA	5.233	211	HYDRO_G	20	600	17.75	211	1	HYDRO_G	20

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

Table 5.3. Tripping action details for Test 3

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	Details
DISTR1	5.1	201	202	1	• Distance relay (DISTR1) at Circuit 1 from Bus 201 to Bus 202 is activated as Zone 1 and its timer starts at *t* = 5 s. • Zone 1 timer times out at *t* = 5.067 s; self-trip breaker timer starts at the same time. • Circuit 1 from Bus 201 to Bus 202 is tripped at *t* = 5.1 s. • Channel plots for Bus 201 and 202 are shown in Figure 5.5.
VTGTPA	TimeOut	Bus	BusName	BuskV	Details
VTGTPA	5.233	211	MINE_G	20	VTGTPA at Bus 211: • Pickup timer starts at *t* = 5.00 s. • Breaker timer starts at *t* = 5.15 s. • Breaker timer times out at time *t* = 5.233 s. • Channel plot for Bus 211 is shown in Figure 5.6 • Voltage at Bus 211 starts to recover after tripping and reaches a steady state around 60 s.

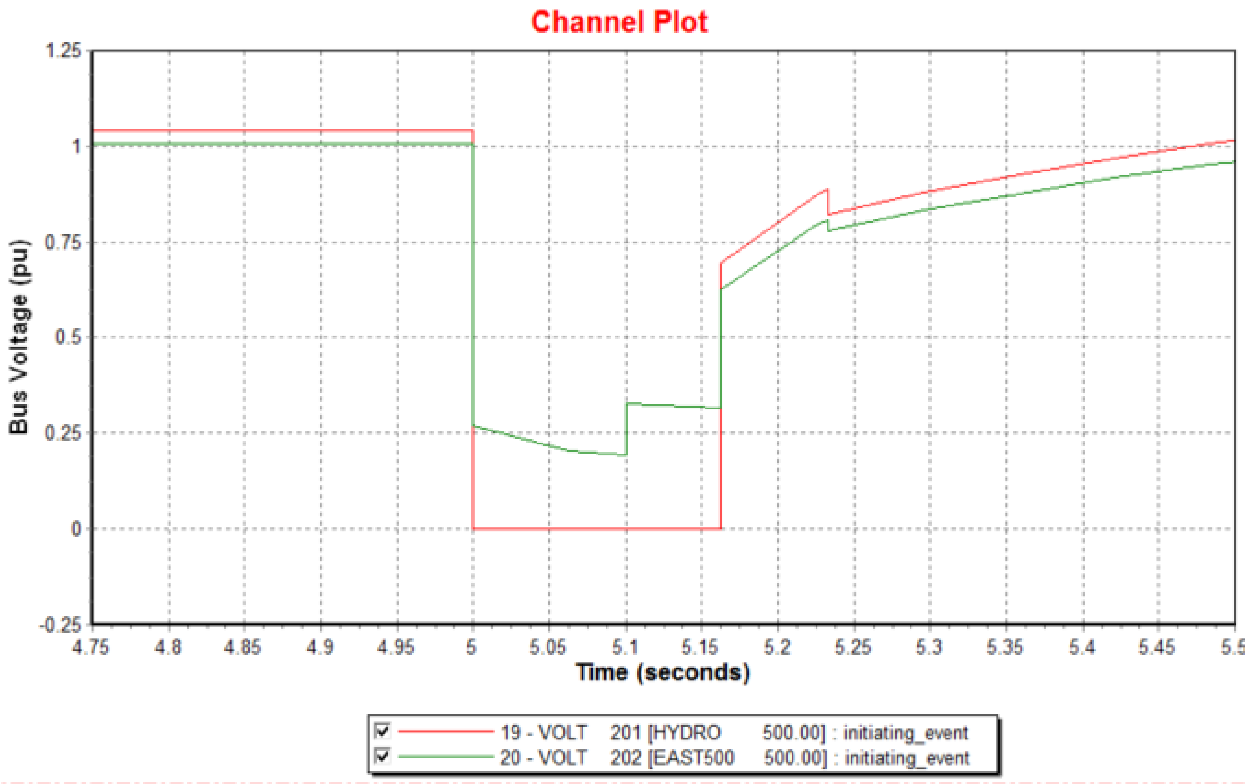


Figure 5.5. Channel plot for voltages at Buses 201 and 202

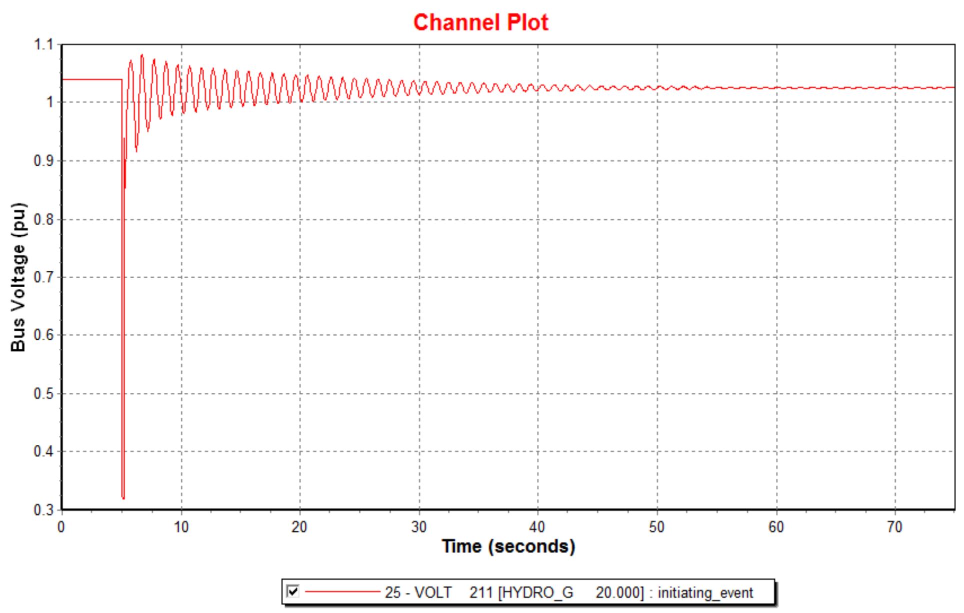


Figure 5.6. Channel plot for voltage at Bus 211

5.1.4 Example 4 in “savnw” System: 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 5.4 shows a summary.

Table 5.4. Relay trips summary for Test 4

Relay Type DISTR1	Time Out	Bus from	Busto	ckt						
DISTR1	5.1	151	152	1						
DISTR1	5.1	151	152	2						
VTGTPA	Time Out	Bus	BusName	BuskV	Pgen (MW)	Qgen (MVAr)	GenBus	GenID	GenName	GenkV
VTGTPA	5.2333	101	NUC-A	21.6	750	81.19	101	1	NUC-A	21.6
VTGTPA	5.2333	102	NUC-B	21.6	750	81.19	102	1	NUC-B	21.6
FRQTPA	Time Out	Bus	BusName	BuskV	Pgen	Qgen	GenBus	GenID	GenName	GenkV
FRQTPA	6.3583	3018	CATDOG_G	13.8	100	80	3018	1	CATDOG_G	13.8
FRQTPA	6.3666	206	URBGEN	18	800	600	206	1	URBGEN	18
FRQTPA	6.3666	3011	MINE_G	13.8	258.66	104.04	3011	1	MINE_G	13.8
FRQTPA	6.3791	211	HYDRO_G	20	600	17.75	211	1	HYDRO_G	20

The sequence of tripping is shown in Figure 5.7. Tripping 1 is due to a distance relay, Tripping 2 is due to undervoltage at Generators 101 and 102, and the remaining tripping actions 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 5.5. Simulation result plots are shown in Figure 5.8 and Figure 5.9.

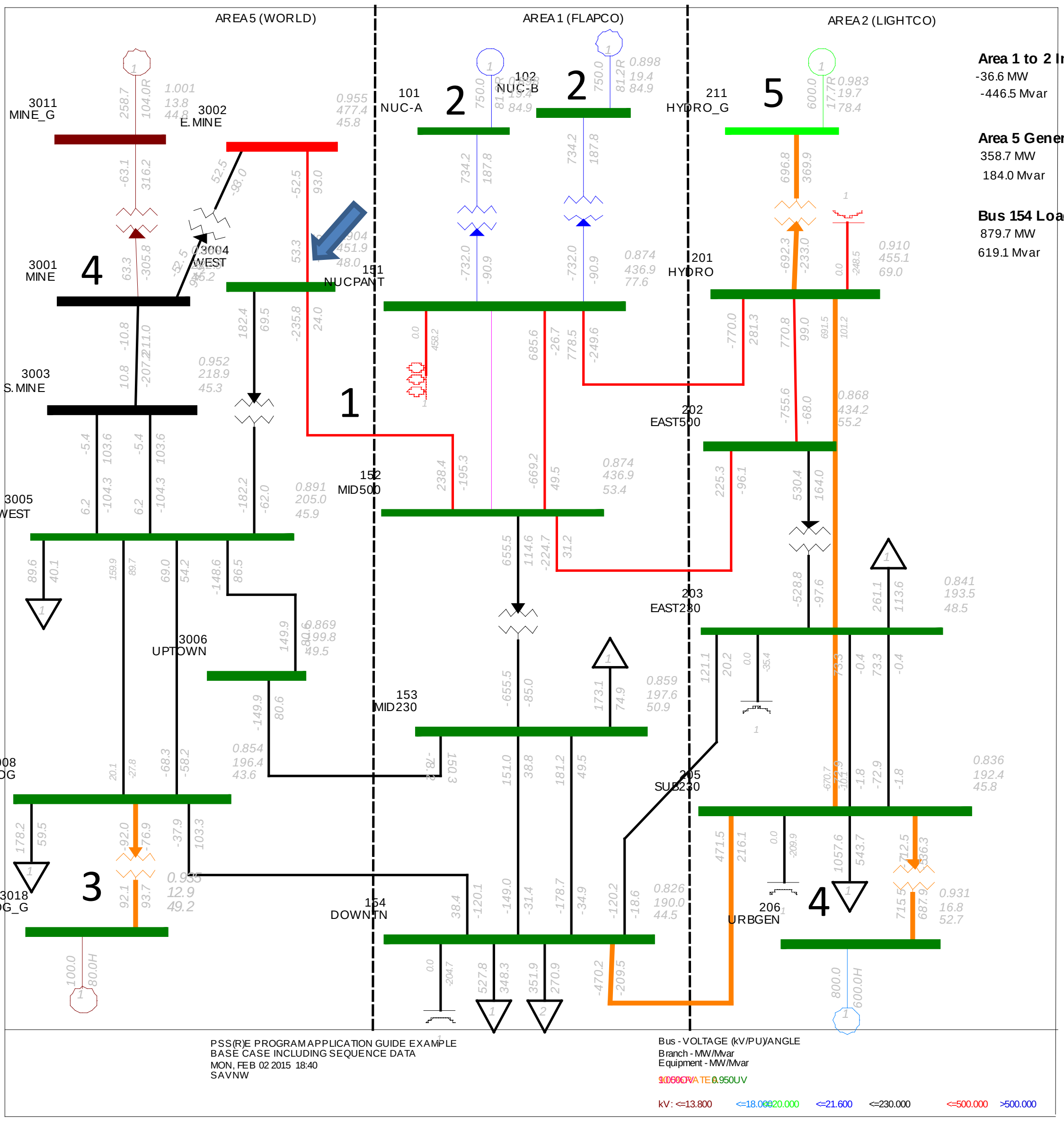


Figure 5.7. One-line diagram of test system to show tripping sequence

Table 5.5. Tripping action details for Test 4

Relay Type					
DISTR1	TimeOut	Busfrom	Busto	ckt	Details
DISTR1	5.1	151	152	1	• Distance relay (DISTR1) at Circuit 1 from Bus 151 to Bus 152 and relay at Circuit 2 from Bus 151 to Bus 152 are activated as Zone 1 and their timers start at *t* = 5 s. • Zone 1 timer times out at *t* = 5.067 s; self-trip breaker timer starts at the same time. • Circuit 1 from Bus 151 to Bus 152 and Circuit 2 from Bus 151 to Bus 152 are tripped at *t* = 5.1 s.
DISTR1	5.1	151	152	2	
VTGTPA	TimeOut	Bus	BusName	BuskV	Details
VTGTPA	5.2333	101	NUC-A	21.6	VTGTPA at Buses 101 and 102 • Pickup timer starts at *t* = 5.000 s. • Breaker timer starts at *t* = 5.150 s. • Breaker timer times out at time *t* = 5.2333 s. • Channel plots for Buses 101 and 102 are shown in Figure 5.8.
VTGTPA	5.2333	102	NUC-B	21.6	
FRQTPA	TimeOut	Bus	BusName	BuskV	Details
FRQTPA	6.3583	3018	CATDOG_G	13.8	FRQTPA at Bus 3018 • Pickup timer starts at *t* = 6.271 s. • Breaker timer starts at *t* = 6.275 s. • Breaker timer times out at time *t* = 6.358 s.
FRQTPA	6.3666	206	URBGEN	18	FRQTPA at Buses 206 and 3011 • Pickup timer starts at *t* = 6.279 s. • Breaker timer starts at *t* = 6.283 s. • Breaker timer times out at time *t* = 6.3666 s.
FRQTPA	6.3666	3011	MINE_G	13.8	
FRQTPA	6.3791	211	HYDRO_G	20	FRQTPA at Bus 3011 • Pickup timer starts at *t* = 6.292 s. • Breaker timer starts at *t* = 6.296 s. • Breaker timer times out at time *t* = 6.379 s. • Channel plots for speeds of Machines 3018, 206, 3011, and 211 are shown in Figure 5.9.

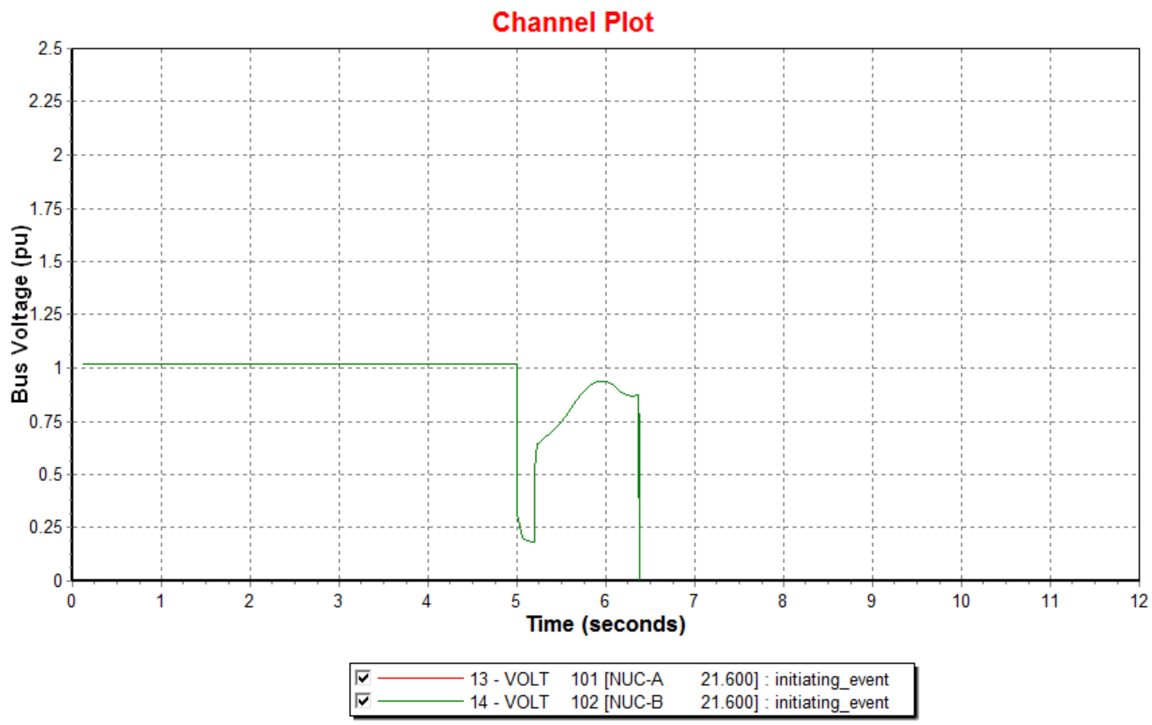


Figure 5.8. Channel plots for voltages at Buses 101 and 102

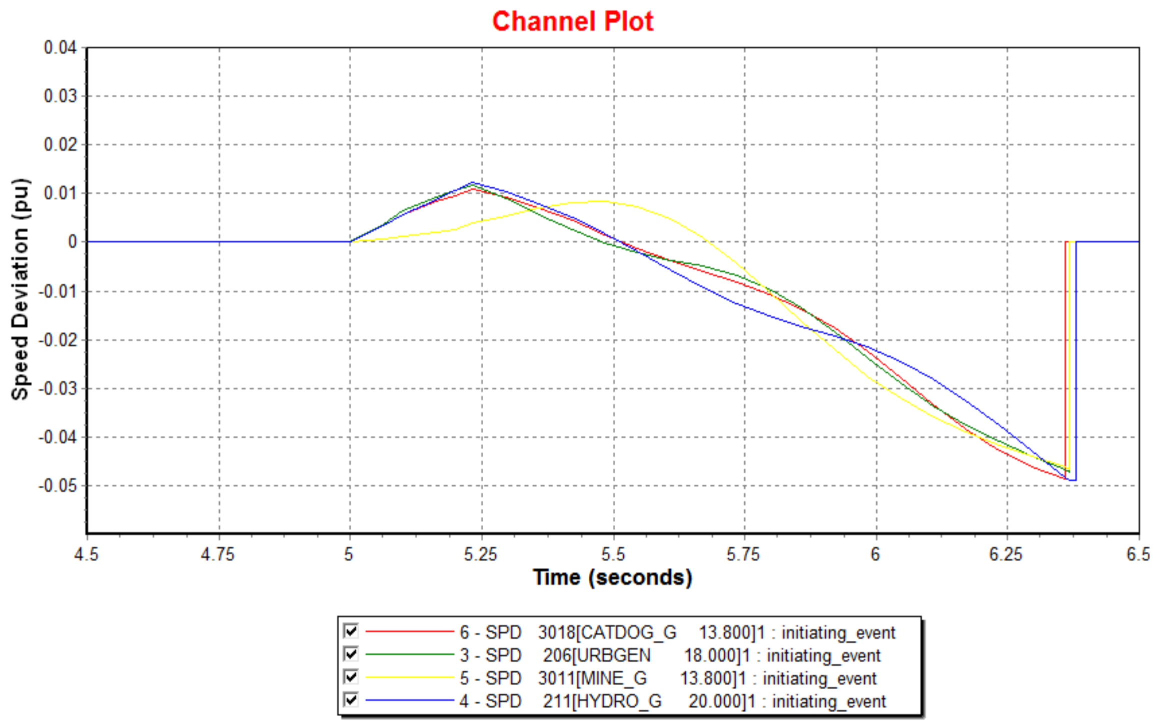


Figure 5.9. Channel plots for speeds of Machines 3018, 206, 3011, and 211

5.1.5 Example 5 in “savnw” System: Activation of an SPS/RAS

In this example, a bus fault that lasts for six cycles is introduced at Bus 203, which is then tripped to isolate the fault. This is an extreme event that has the potential to trigger an SPS/RAS. The fault was introduced at *t* = 5 seconds and the bus was isolated after six cycles, along with a line trip during the dynamic simulation. No relay tripping is observed during the dynamic simulation. A graph of the simulation result is shown in Figure 5.11.

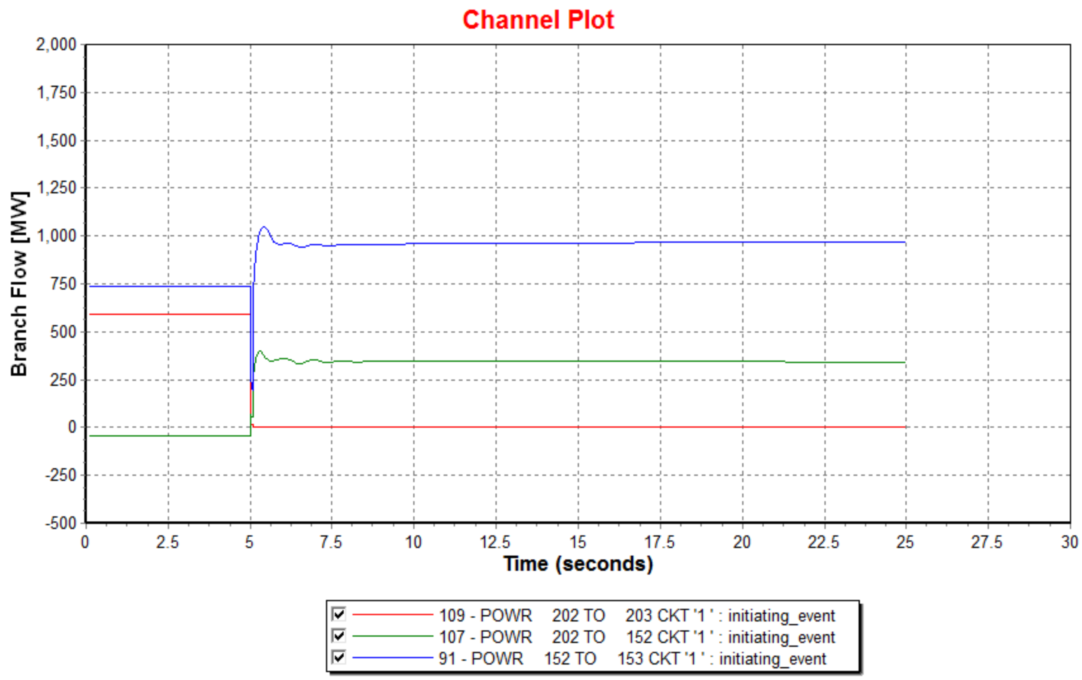


Figure 5.10. Example 5 for savnw system: branch flows

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 is performed to trigger this cascading event.

No other trippings are observed during the dynamic simulation where an SPS/RAS event is triggered. The line overloads observed on the system are below 130% of Rate A and no voltage violations below 0.9 pu are observed. No corrective action is required for this contingency with these protection settings. The sequence of DCAT actions that are performed for this contingency is shown in Figure 5.11. This contingency results in no tripping actions, generation loss, or load loss, as given in Table 5.6.

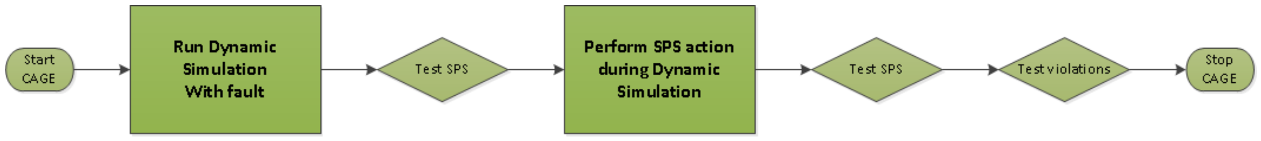

Start CAGE
Run Dynamic Simulation With fault
Test SPS
Perform SPS action during Dynamic Simulation
Test SPS
Test violations
Stop CAGE

Figure 5.11. Sequence of events performed by DCAT for Example 5 in “savnw” system

Table 5.6. Generation and load loss summary for Example 5 in “savnw” system

Load loss (MW)	0
No. of total tripping actions	0
No. of SPSs/RASs triggered	1
No. of overloaded lines	0
Corrective actions	None

5.2 Test System 2: Polish System

The Polish system has 3120 buses and 505 generators. The dynamics database includes generator, governor, stabilizer, and exciter models for generators, load dynamic models for loads, etc. Generator protection relays (FRATPAT, VTGTPAT), transmission protection relays (DISTR1), and load shedding relays (UVUFBLU) are added to the existing relay protection models in the Polish case. Dynamic simulation is performed to accurately simulate the effect of major contingencies and capture the cascading tripping actions due to tripping of multiple protection relays that are not captured in traditional dynamic and steady-state simulation.

Four different examples are provided for this system. Table 5.7 shows a summary of the four examples for the Polish system.

Table 5.7. Comparison of Polish simulation examples

	Example 1	Example 2	Example 3	Example 4
	Bus Fault	Line Fault	Line Fault with Zone 1 maloperation	Generation fault
Number of dynamic simulations	1	1	2	Multi-processing modules were used to initiate several instances of PSS/E
TOTAL NUMBER OF TRIPS	3	2	21	
Number of generators tripped	3	0	20	
Total lines tripped by relay	0	1	1	
Total generation lost	430 MW	0 MW	1657 MW	
Total load lost	0 MW	0 MW	212 MW	
Violations	No Violations	No violation	No Violations	
Corrective actions	NA	NA	NA	
RAS/SPS	No	No	No	

5.2.1 Example 1: Bus Fault in Polish System

Contingency: In this example, a three-phase fault is applied at Bus 133 in the Polish system. Figure 5.13 is a partial one-line diagram of the Polish system. The fault is applied at *t* = 5 s and it is cleared after five cycles in the dynamic simulation. Dynamic simulation is performed to accurately simulate the effect of major contingencies and capture the cascading tripping actions due to trippings of multiple protection relays that are not captured in traditional dynamic and steady-state simulation. The fault is cleared by tripping all the substation transformers and both high-voltage and low-voltage buses of substations that are connected with Bus 133. The sequence of events after the fault is listed in Table 5.8.

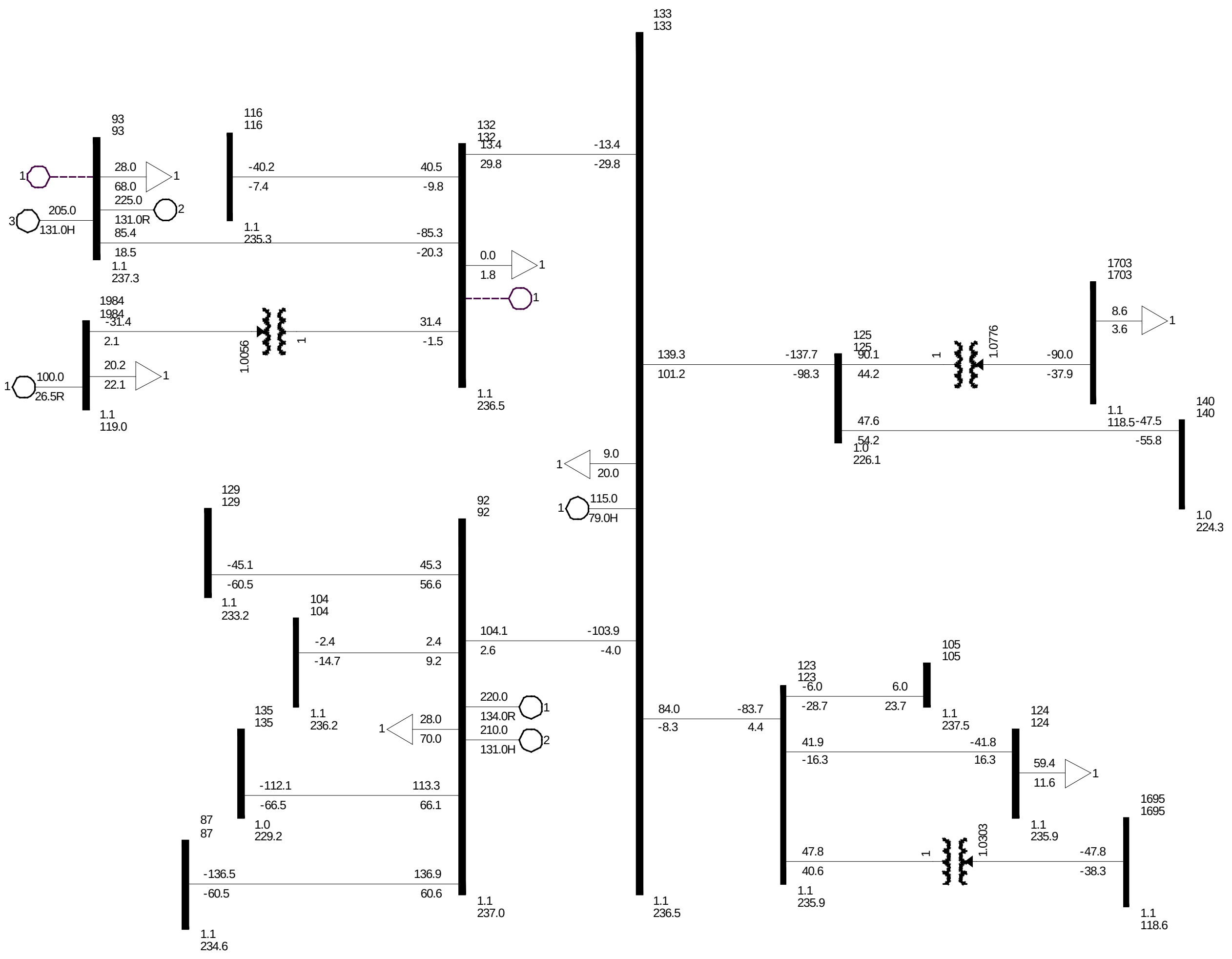


Figure 5.12. Partial one-line diagram of Bus 133 of the Polish system

<u>Key findings:</u> Figure 5.14 shows the one-line diagram of Bus 133 after the fault. The dotted lines indicate tripped lines. During the dynamic simulation, low- or high-voltage ride-through “VTGTPAT” relay trips three generators. The generator units at Buses 1588 and Bus 1589 are tripped due to overvoltage (dv = 0.10), while the generator unit at Bus 3118 is tripped due to overvoltage (dv = 0.2). The total generation lost is 430 MW. Figure 5.15 shows a plot of the terminal voltage of these generator units.

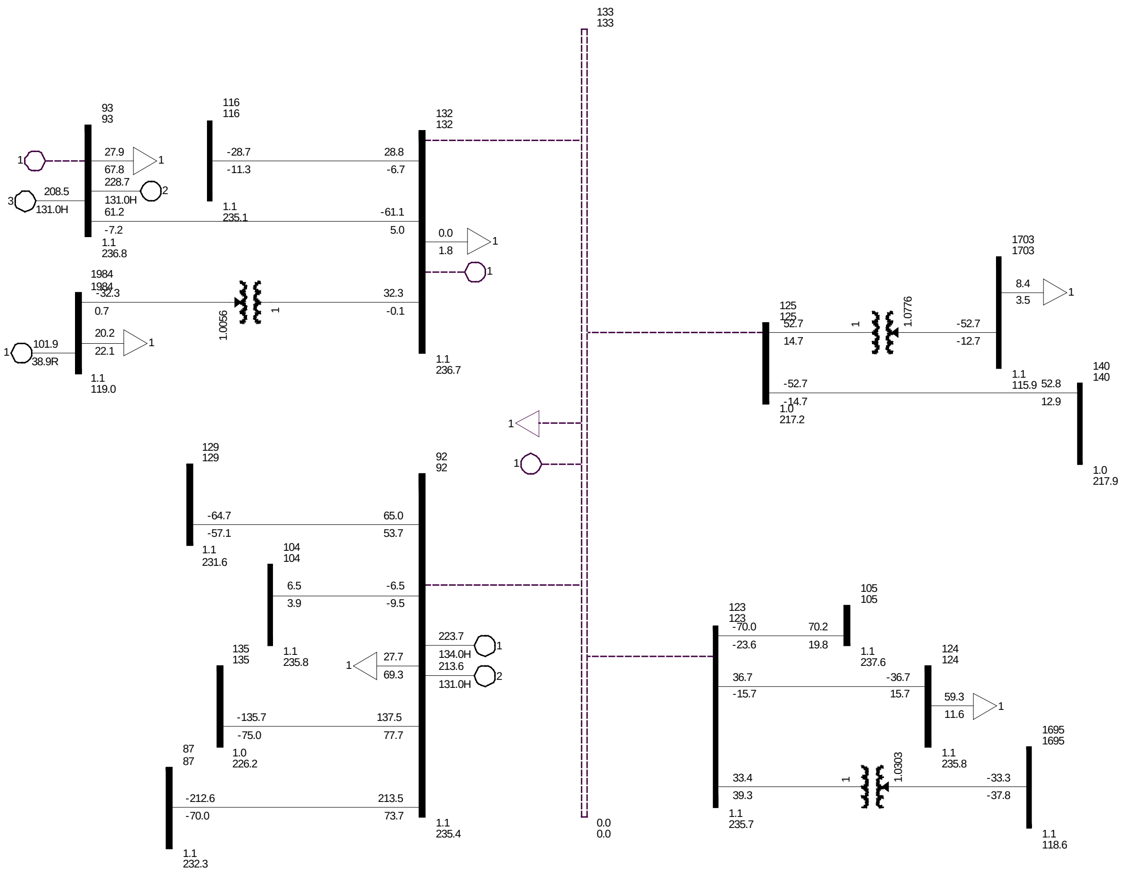


Figure 5.13. Partial one-line diagram of Bus 133 of the Polish system immediately following a fault

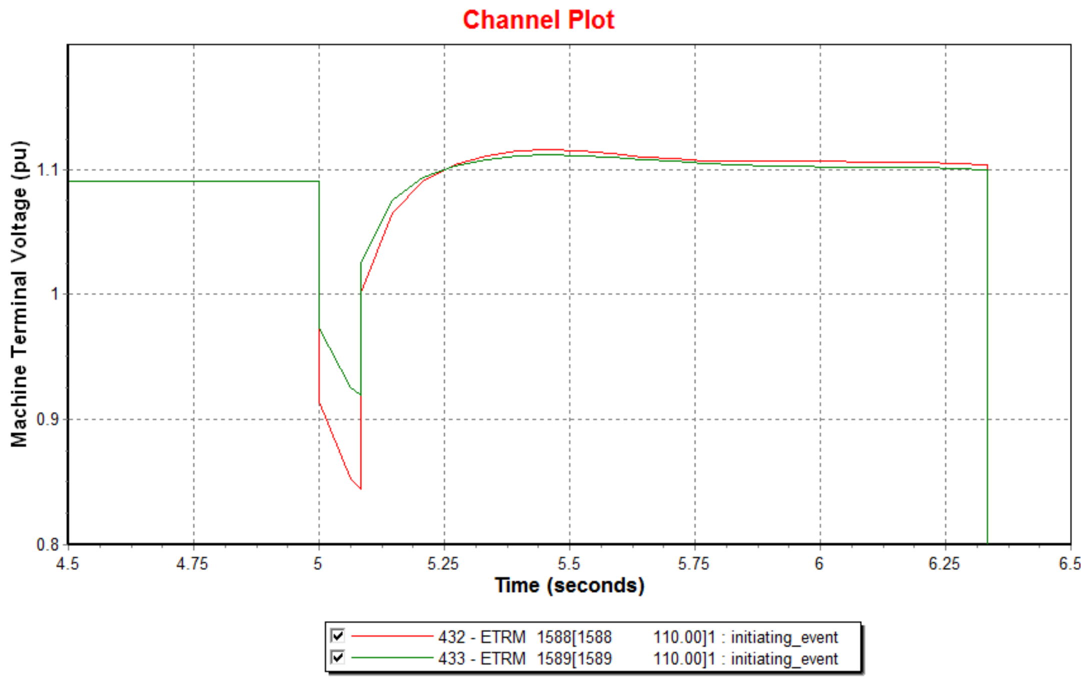


Figure 5.14. Terminal voltage of the tripped generator units in Example 1 in Polish System

The line overloads observed on the system are below 130% of Rate A and no voltage violations below 0.9 pu are observed; therefore, no corrective action is needed. A summary of the sequence of relay trippings that are observed during the dynamic simulation is shown in Table 5.8.

Table 5.8. Tripping summary during dynamic simulation for Example 1 in Polish System

No. of generators disconnected	3
Generator outage (MW)	430
Load shedding (MW)	0

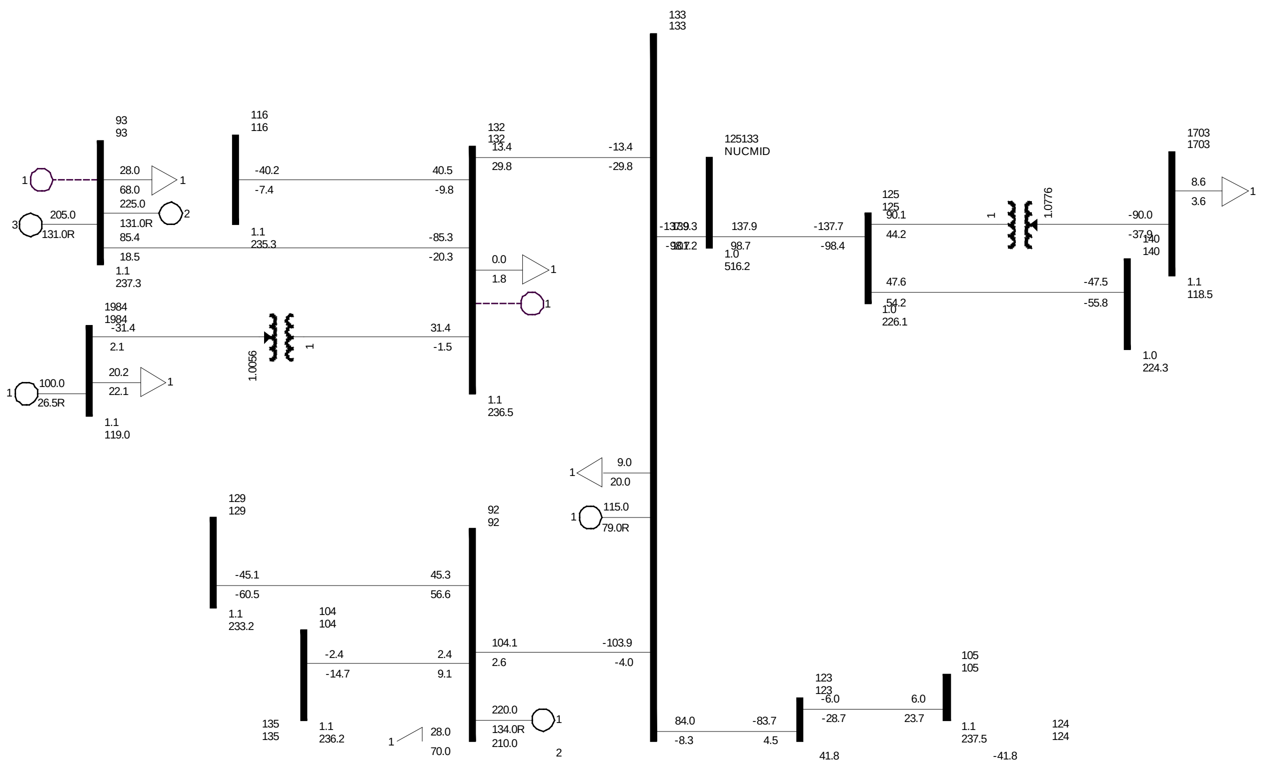
5.2.2 Example 2: Line Fault in Polish System

A line fault is applied to the branch from Bus 125 to Bus 133 at a distance of 10% from Bus 125 at time *t* = 5 seconds, as shown in Figure 5.15a. The fault is set to remain for 10 cycles. Generator protection relays, load shedding relays, and transmission protection relays (Zone 1 and Zone 2 protection) are added to the existing protection models in the Polish system case. Zone 1 settings initiate circuit breaker tripping after six cycles of the fault, and the Zone 2 fault setting trips the line after 20 cycles.

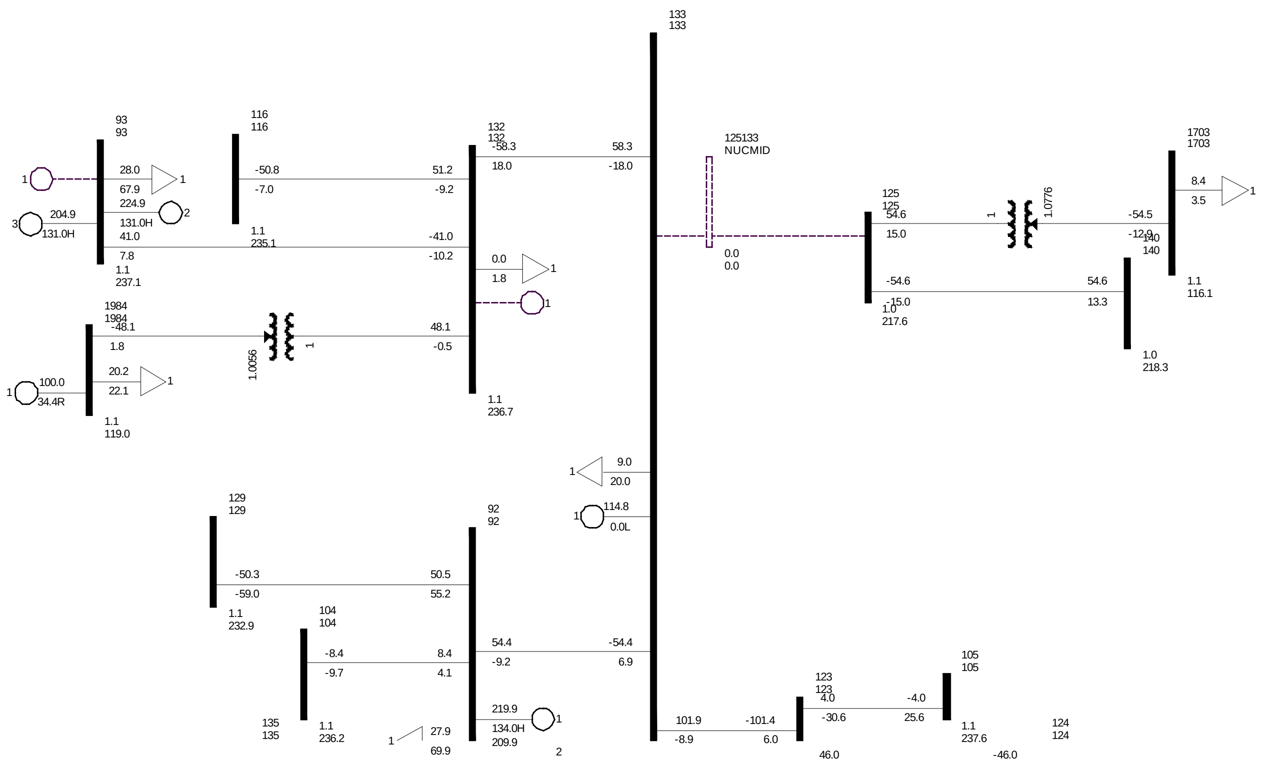
A line fault is applied on one of the lines connected to Bus 133 at a distance of 90% from it. Distance relays are modeled on both ends of the line with an ability to send a transfer trip to the other end upon sensing a Zone 1 fault. To model a fault in PSS/E at any location in a transmission line other than the two line ends, a fictitious node needs to be added. In this test, to model a fault in the line connecting Buses 125 and 133 that is located at a distance of 0.1 pu of total line length from Bus 125, a new fictitious node (125133) is added between Buses 125 and 133. Figure 5.15a 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. The dotted lines in Figure 5.15b represent tripped lines.

The bus fault is introduced at the fictitious bus (125133) at *t* = 5 s, and the 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. The distance relay (DISTR1) at Circuit 1 from Bus 125 to Bus 125133 is activated as Zone 1 and its timer starts at *t* = 5 s.
2. The distance relay (DISTR1) at Circuit 1 from Bus 133 to Bus 125133 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 the transfer trip and breaker trip timers start at the same time.
4. Circuit 1 from Bus 125 to Bus 125133 trips at *t* = 5.05 s and the transfer trip timer 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 Bus 133 to Bus 125133) trips at the same time (*t* = 5.05 s), and soon thereafter the two voltages start to recover.



(a)



(b)

Figure 5.15. One-line diagram of Bus 133 (a) before and (b) after the fault

A summary of the sequence of relay trippings that are observed during the dynamic simulation is shown in Table 5.9.

Table 5.9. Tripping sequence for Example 2 in Polish System

Relay Type				
DISTR1	TimeOut	Busfrom	Busto	ckt
DISTR1	5.1	125	125133	1
DISTR1	5.1	133	125133	1

A summary of the sequence of relay trippings that are observed during the dynamic simulation is shown in Table 5.10.

Table 5.10. Tripping summary during dynamic simulation for Example 2 in Polish System

No. of generators disconnected	0
Generator outage (MW)	0
Load shedding (MW)	0
No. of branches disconnected by relay	1

5.2.3 Example 3: Line Fault with Zone 1 Maloperation in Polish System

This example is similar to Example 2, but here it is assumed that Zone 1 protection has failed to trip the line from Bus 125 to Bus 133. To mimic this maloperation, the Zone 1 pickup time on this line is delayed. It is set to 0.83333 seconds (50 cycles). Zone 2 is set to operate after 20 cycles. With these settings, the relay on that line sees the fault as a Zone 2 fault. The sequence of events after the fault is listed in Table 5.11.

Table 5.11. Relay trips summary of Test 8

Relay Type										
DISTR1	TimeOut	Busfrom	Busto	ckt						
DISTR1	5.333	125	125133	1						
VTGTPA	TimeOut	Bus	BusName	BuskV	Pgen (MW)	Qgen (Mvar)	GenBus	GenID	GenName	GenkV
VTGTPAT	5.3833	1447	1447	110	17	0	1447	1	1447	110
VTGTPAT	5.3833	1448	1448	110	17	0	1448	1	1448	110
VTGTPAT	5.3833	1449	1449	110	17	0	1449	1	1449	110
VTGTPAT	6.45	1588	1588	110	215	55.84	1588	1	1588	110
VTGTPAT	6.4541	1589	1589	110	215	75.75	1589	1	1589	110
VTGTPAT	6.4541	1668	1668	110	125	50.17	1668	1	1668	110
VTGTPAT	6.4541	1668	1668	110	125	50.17	1668	2	1668	110
VTGTPAT	6.475	1982	1982	110	123	0	1982	1	1982	110

VTGTPAT	6.475	1982	1982	110	123	0	1982	2	1982	110
VTGTPAT	6.4875	93	93	220	225	131	93	2	93	220
VTGTPAT	6.4875	93	93	220	225	131	93	3	93	220
VTGTPAT	6.4875	133	133	220	115	79	133	1	133	220
VTGTPAT	6.4875	1302	1302	110	15	0	1302	2	1302	110
VTGTPAT	6.4875	1984	1984	110	100	26.51	1984	1	1984	110
VTGTPAT	6.4916	92	92	220	220	134	92	1	92	220
VTGTPAT	6.4916	92	92	220	220	134	92	2	92	220
VTGTPAT	6.5041	1983	1983	110	100	38.98	1983	1	1983	110
VTGTPAT	6.5208	1769	1769	110	43	30.3	1769	1	1769	110
VTGTPAT	6.5291	1993	1993	110	110	5.19	1993	2	1993	110
VTGTPAT	26.0019	3118	3118	400	0	−114.08	3118	1	3118	400
AC corrective actions are applied										
Dynamic Simulation 2										
	1.000082			Trip highest overloaded line; OVERLOADING: FROM 143 TO 126 BRANCH ID 1 WITH OVERFLOW 100.039550781% IS GOING TO BE REMOVED						
	30			Check for steady state: Reached						
AC corrective actions are applied; no overloaded lines										
End of DCAT simulation										

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

Since the Zone 1 protection on the faulted line fails, the fault is not cleared immediately. It is cleared after 20 cycles when the Zone 2 distance protection relay detects the fault. This maloperation causes a cascaded tripping of several units. The contingency described in this scenario causes a total of 20 generators and one line to trip. This is an unusual contingency scenario, but DCAT can accurately model it, unlike other existing tools. The total generation lost is 1657 MW. Figure 5.18 shows terminal voltages of three generators that are tripped during the course of generator trippings. During the dynamic simulation, the under/overvoltage generator-trip relay timers (at Generators 1447, 1588, and 93) of these generators start and trip after the fault is introduced. Units 1448, 1588, and 93 are tripped due to voltage violations.

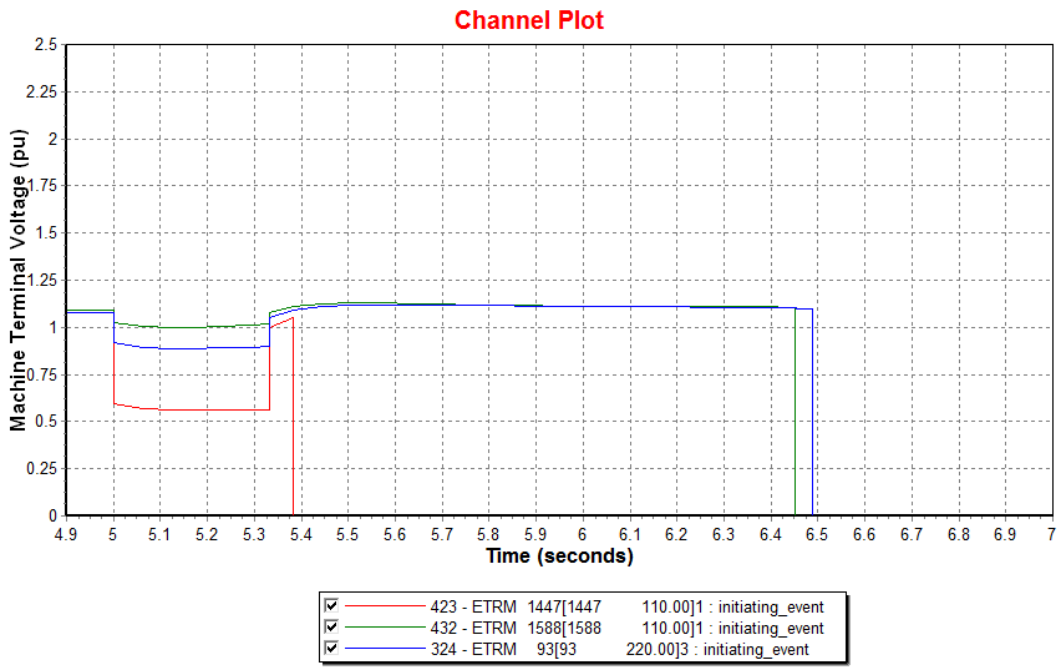


Figure 5.16. Terminal voltage of the tripped generator units for Example 3 in Polish system

5.2.3.1 Dynamic Simulation 1

- The system reaches steady state after 30 seconds. AC corrective actions are applied to the solved post-dynamic power flow case; the AC corrective action results are shown in Table 5.12.

Table 5.12. Results of AC corrective actions for Dynamic Simulation 1

BUS	GEN-INI	GEN-NEW	GEN-ADJ
56 56 220.00	219.5	225	5.5
57 57 220.00	436.7	500	63.3
68 68 220.00	223.4	225	1.6
69 69 220.00	446.8	500	53.2
121 121 220.00	123.7	70	−53.7
122 122 220.00	122.9	70	−52.9
143 143 220.00	447.8	490	42.2
240 240 220.00	238.4	250	11.6
876 876 110.00	121.8	179	57.2
922 922 110.00	218.2	225	6.8
985 985 110.00	17.6	30	12.4
990 990 110.00	0.9	2	1.1
1011 1011 110.00	202.1	225	22.9
1057 1057 110.00	9.7	12	2.3

BUS	GEN-INI	GEN-NEW	GEN-ADJ
1086 1086 110.00	117	125	8
1087 1087 110.00	117.8	125	7.2
1132 1132 110.00	64.7	25.5	−39.1
1164 1164 110.00	58.6	11	−47.6
1188 1188 110.00	59.7	42	−17.7
1189 1189 110.00	59.7	42	−17.7
1263 1263 110.00	117.9	71.8	−46.1
1268 1268 110.00	11.1	17	5.9
1429 1429 110.00	56.7	20	−36.7
1476 1476 110.00	94.5	37.2	−57.3
1546 1546 110.00	11.5	4.6	−6.9
1547 1547 110.00	11.8	13.5	1.7
1656 1656 110.00	172.7	90	−82.7
1658 1658 110.00	127.7	79	−48.7
1681 1681 110.00	108.2	119	10.8
1742 1742 110.00	16.4	62	45.6
1954 1954 110.00	67.9	50.5	−17.3
2144 2144 110.00	219.2	225	5.8
2146 2146 110.00	218.7	225	6.3
3103 3103 110.00	137.3	150	12.7
3119 3119 32.000	63.5	112	48.5
BUS	**LOAD-INI**	**LOAD-NEW**	**LOAD-ADJ**
38 38 400.00	56.6	56.2	0.4
1200 1200 110.00	9.1	0	9.1
1201 1201 110.00	14.8	7.1	7.8
1203 1203 110.00	14.9	0	14.9
1236 1236 110.00	4.5	0	4.5
1237 1237 110.00	3.5	0	3.5
1412 1412 110.00	18.2	0	18.2
1489 1489 110.00	1.9	0	1.9
1555 1555 110.00	2.8	0	2.8
1556 1556 110.00	0.8	0	0.8
1585 1585 110.00	1.9	0	1.9
1642 1642 110.00	6.6	0	6.6
1643 1643 110.00	4.6	0	4.6

BUS	GEN-INI	GEN-NEW	GEN-ADJ
1696 1696 110.00	7.1	0	7.1
1697 1697 110.00	1.8	0	1.8
1794 1794 110.00	2.8	0	2.8
1795 1795 110.00	2.1	0	2.1
1812 1812 110.00	1.1	0	1.1
1813 1813 110.00	5.6	0	5.6
1969 1969 110.00	3.7	0	3.7
1970 1970 110.00	10.2	0	10.2
1974 1974 110.00	3.5	0	3.5
1975 1975 110.00	3	0	3
1983 1983 110.00	13.3	0	13.3
2102 2102 110.00	0.6	0	0.6
2163 2163 110.00	1.9	0	1.9
2170 2170 110.00	6.4	0	6.4
2171 2171 110.00	5.5	0	5.5

- Overloaded lines are checked after the OPF and corrective actions are applied; two lines are found to have flow violations:

*OVERLOADING: FROM 143 TO 126 BRANCH ID 1 OVERLOAD 100.039550781%

*OVERLOADING: FROM 143 TO 126 BRANCH ID 1 WITH OVERFLOW 100.039550781% IS GOING TO BE REMOVED.

5.2.3.2 Dynamic Simulation 2

- A second dynamic simulation is run with the highest overloaded line tripped.
- After the second dynamic simulation reaches steady state, AC corrective actions are applied on the solved post-dynamic power flow case; the AC corrective action results are shown in Table 5.13.

Table 5.13. Results of AC corrective actions for Dynamic Simulation 2

BUS	GEN-INI	GEN-NEW	GEN-ADJ
56 56 220.00	224.8	225	0.2
57 57 220.00	499.3	500	0.7
58 58 400.00	316.1	491.1	175
59 59 400.00	315.4	490.4	175
68 68 220.00	224.8	225	0.2
69 69 220.00	494.3	500	5.7

BUS	GEN-INI	GEN-NEW	GEN-ADJ
71 71 400.00	467.7	503.5	35.8
96 96 400.00	781.5	706.5	−75
142 142 220.00	218.3	166.2	−52
143 143 220.00	489.7	364.7	−125
240 240 220.00	249.8	250	0.2
251 251 110.00	22.1	23	0.9
268 268 110.00	2.3	0	−2.3
301 301 110.00	13.7	65	51.3
302 302 110.00	4.3	5	0.7
517 517 110.00	191.8	211.1	19.3
740 740 110.00	135	207	72
741 741 110.00	118.3	205	86.7
764 764 110.00	1.5	51	49.5
772 772 110.00	27.4	70	42.6
773 773 110.00	86.5	97	10.5
876 876 110.00	147.8	179.2	31.4
922 922 110.00	224.8	225	0.2
1011 1011 110.00	224.9	225	0.1
1067 1067 110.00	7.5	8	0.5
1086 1086 110.00	124.9	125	0.1
1087 1087 110.00	124.9	125	0.1
1164 1164 110.00	11.5	62.4	51
1188 1188 110.00	42.3	42	−0.3
1189 1189 110.00	42.3	42	−0.3
1263 1263 110.00	71.7	32	−39.7
1289 1289 110.00	391.5	216.5	−175
1290 1290 110.00	391.5	216.5	−175
1476 1476 110.00	37.6	21	−16.6
1546 1546 110.00	4.7	3	−1.7
1648 1648 110.00	59.6	0	−59.6
1681 1681 110.00	118.9	119	0.1
1742 1742 110.00	21.4	62	40.6
1954 1954 110.00	50.9	45.8	−5.1
2144 2144 110.00	224.8	135	−89.8
2146 2146 110.00	224.8	225	0.2

BUS	GEN-INI	GEN-NEW	GEN-ADJ
2256 2256 110.00	3	−48.2	−51.2
2267 2267 110.00	3	0	−3
2271 2271 110.00	1.8	0	−1.8
2357 2357 110.00	2.7	0	−2.7
2370 2370 110.00	2.4	1	−1.4
2383 2383 110.00	0.6	0	−0.6
2495 2495 110.00	2.2	1.1	−1.1
2522 2522 110.00	0.4	0	−0.4
2591 2591 110.00	0.5	0	−0.5
2621 2621 110.00	3.2	−48	−51.2
2644 2644 110.00	5.6	2	−3.6
2684 2684 110.00	8	0	−8
2685 2685 110.00	7.4	2	−5.4
3103 3103 110.00	149.9	150	0.1
3119 3119 32.000	111.8	112	0.2
BUS	LOAD-INI	LOAD-NEW	LOAD-ADJ
92 92 220.00	25.1	13.4	11.7
1201 1201 110.00	7.1	2.5	4.6
1297 1297 110.00	8.6	0	8.6
1298 1298 110.00	6.6	0	6.6
1770 1770 110.00	2.3	0	2.3
1772 1772 110.00	6.3	0	6.3
1773 1773 110.00	7.8	0	7.8
2070 2070 110.00	2.6	0	2.6
2071 2071 110.00	2.4	0	2.4
2151 2151 110.00	9.7	0	9.7
2152 2152 110.00	4.9	0	4.9

Overloaded lines are checked again after the corrective actions are applied; this time NO line is found with a flow violation.

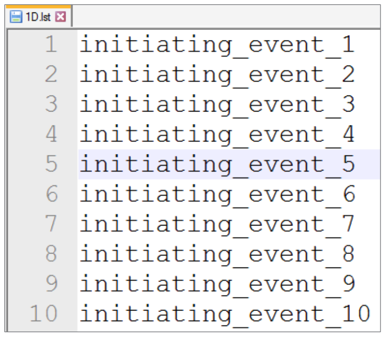
The DCAT simulation is terminated, because there were no violations at the end of dynamic simulation. Most of the tripping actions happen immediately following an extreme event, which is captured in the dynamic simulation. A summary of the sequence of relay trippings that are observed during the dynamic simulation is shown in Table 5.14. This contingency resulted in a total of 21 tripping actions. All the generating units are tripped due to voltage violations.

Table 5.14. Tripping summary during dynamic simulation for Example 3

No. of generators disconnected	21
Generator outage (MW)	1657
Load shedding/curtailment (MW)	212 (load curtailed during corrective actions)
No. of branches disconnected by relay	1

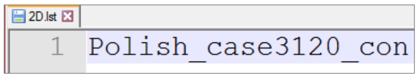
5.2.4 Example 4: Generation Fault – Polish System

In Example 4, MPjobs used multiple processing modules to initiate several instances of PSS/E, to run the same PSS/E script on different data sets. In short, MPjobs conducted dynamic simulation in multiple single-generator and line-contingency scenarios in parallel by using multiple processors to reduce simulation time significantly. MPjobs is modularized into the DCAT GUI that will control multiple runs in parallel, calling the function of MainDCAT. In order to initiate several contingency simulations using MPjobs, all the .idv file names (termed “Xfile” in configuration) should be listed and saved in the same folder as the 1D list file, as shown in Figure 5.17. A list of save/snp files showing different base-case names (termed “Yfile” in configuration), should be listed and saved in the same folder as the 2D list file, as shown in Figure 5.18. MPjobs will conduct the DCAT runs for cases using each of the .idv files. The results for each case will be saved in a separate folder named after the .idv file name.



```
initiating_event_1
initiating_event_2
initiating_event_3
initiating_event_4
initiating_event_5
initiating_event_6
initiating_event_7
initiating_event_8
initiating_event_9
initiating_event_10
```

Figure 5.17. 1D list file



```
Polish_case3120_con
```

Figure 5.18. 2D list file

6.0 Introduction to Details of Python Modules in the DCAT Package

This section introduces details of various Python modules in the DCAT package. As described in Section 3.0, modules are categorized in four packages. DCAT is highly modularized and most of the modules can be used independently. The independence of the modules supports future development and the feasibility of user contribution. Modification of a certain step could be restricted to the corresponding module. Extension of a model is achievable by adding a new module to an existing module package or a new package. Documentation is organized through extensive comments inside the code and the example configuration files.

6.1 "Configuration.py" in Package "DataReader"

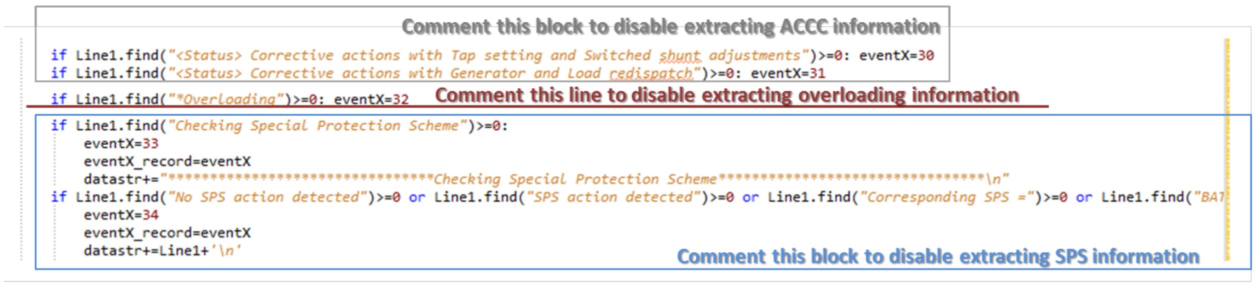
The configuration module controls the input/output interface of DCAT by defining a configuration class that stores all the related configuration information. The user communicates with DCAT through the user-defined configuration file by using this module. The major functions are

1. "ReadINI": reads the .ini file and stores the variables into the configuration class.
2. "ExamineInputFiles": examines the existence of the input files; if the required input files do not exist, error messages will be displayed and DCAT will exit.
3. "WriteINI": writes the stored configuration information in an .ini file, along with the descriptions for all the settings variables listed in the .ini file.

The defined configuration class is the only input item for the main function in "MainDCAT." An example script of how to initialize a configuration class, load data from an .ini file, and assign this configuration class to the main function is shown in Section 7.0.

6.2 "ReadLogFile.py" in Package "PostProcessing"

This Python module defines useful subroutines associated with 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. Users do not need to modify this file during regular use of this DCAT package. Please refer to Figure 6.1 if ACCC information, SPS information, or overloading information is not needed.


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

```
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'
```

Figure 6.1. How to disable extracting some information

Below is an example, in a script:

```
from PostProcessing import ReadLogFile33 as ReadLogFile
ReadLogFile.main1(logfile)
```

This will selectively extract data from log files created by PSS/E dynamic runs and output them into a corresponding "*_relay.csv" file. In the current version, two module files are included for PSS/E Version 33 ("ReadLogFile33") and Version 34 ("ReadLogFile34"). Different versions of PSS/E will generate different formats in log files. Hence, DCAT will automatically use the correct "ReadLogFile" to extract information.

6.3 Module Package "Utils"

6.3.1 "Supportingtools"

This Python module defines many useful subroutines associated with file processing. This module serves as a supporting file and is used by other Python modules. Users do not need to modify this file during the regular use of this DCAT package. The functions in this module can be imported using the following:

```
from Utils.Supportingtools import *
```

6.3.2 "Logging"

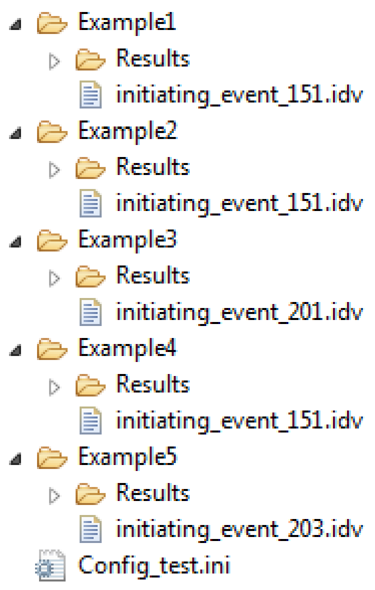
During the execution of DCAT, the code and PSS/E will have many screen outputs. This module defines a logger class that can be used to redirect screen outputs into a log file. It will help the user check screen outputs after completion of the DCAT simulation. For example,

```
from Utils.Logging import Logger # import the Logging module
sys.stdout = Logger() # Initialize the logger
sys.stdout.log = open("*.log", "w") # A log file to save screen outputs
sys.stdout.log.close() # Terminate the logger
sys.stdout = sys.__stdout__ # Return to default logging style
```

6.3.3 "BatchProcessing"

When a series of DCAT simulations are to be executed, this module is newly added as an optional utility function. In batch mode, users can process a series of cases without manual intervention. By calling this module, DCAT will be directed into a folder and determine how many cases need to be simulated sequentially. An example script of how to call this module is shown in Section 6.4. An .ini file and an .idv file are the essentials for each case. The user is required to define the .ini files and .idv files and arrange them in a common folder. Three different arrangements of multiple cases are considered:

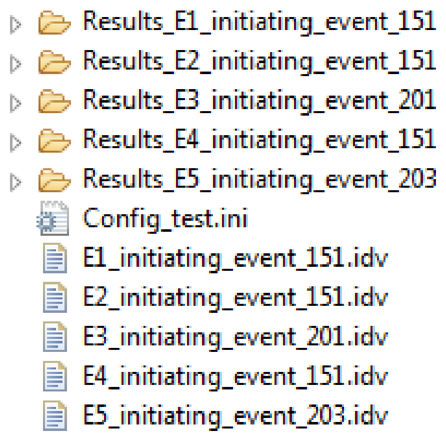
A. A shared configuration file is saved in the common folder; each case has a separate subfolder in this common folder and the .idv file is saved in the case folder. The outputs will be saved in a subfolder in each case folder, as shown in Figure 6.2.



```
Example1
    Results
    initiating_event_151.idv
Example2
    Results
    initiating_event_151.idv
Example3
    Results
    initiating_event_201.idv
Example4
    Results
    initiating_event_151.idv
Example5
    Results
    initiating_event_203.idv
Config_test.ini
```

Figure 6.2. Example arrangement for a batch-processing folder in Situation A

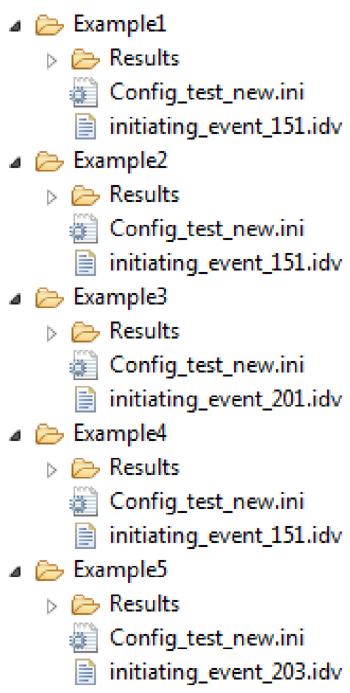
B. A shared configuration file is saved in the common folder; each case has its own .idv file saved directly in the common folder. There are no subfolders to divide the .idv files. DCAT will create an output folder for each case to store results that will be named after the .idv file, as shown in Figure 6.3.



```
Results_E1_initiating_event_151
Results_E2_initiating_event_151
Results_E3_initiating_event_201
Results_E4_initiating_event_151
Results_E5_initiating_event_203
Config_test.ini
E1_initiating_event_151.idv
E2_initiating_event_151.idv
E3_initiating_event_201.idv
E4_initiating_event_151.idv
E5_initiating_event_203.idv
```

Figure 6.3. Example arrangement for a batch-processing folder in Situation A

C. For each case, a separate subfolder is prepared. In a case folder, a case-specified configuration file and the .idv file are stored, as shown in Figure 6.4.



```
◢ Example1
    ▷ Results
      Config_test_new.ini
      initiating_event_151.idv
◢ Example2
    ▷ Results
      Config_test_new.ini
      initiating_event_151.idv
◢ Example3
    ▷ Results
      Config_test_new.ini
      initiating_event_201.idv
◢ Example4
    ▷ Results
      Config_test_new.ini
      initiating_event_151.idv
◢ Example5
    ▷ Results
      Config_test_new.ini
      initiating_event_203.idv
```

Figure 6.4. An example of the arrangement for a batch-processing folder in Situation B

The user can arrange the case files and folders following one of the above formats. DCAT will automatically evaluate and process the batch processing according to the detected cases.

6.4 “MPjobs.py” (Parallel Processing)

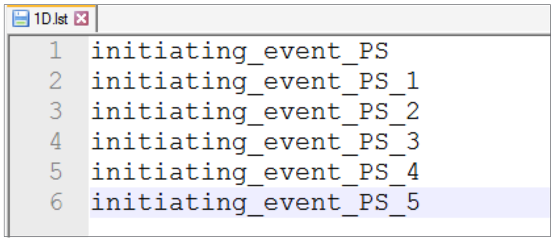
Using a multi-processing module available in Python 2.7 (installed with PSS/E v.33) (Conto 2015), MPjobs activates several instances of PSS/E to run the same PSS/E script on different data sets. In short, MPjobs conducts dynamic simulation in multiple different contingency scenarios (i.e., different .idv files) in parallel by using multiple processors (central processing units [CPUs]) to save significant simulation time. For example, one contingency case costs about 100 seconds, so when only one processor is used to run eight scenarios, the whole process will cost 100 × 8 seconds. Using MPjobs with an 8-CPU computer, the eight cases will run on the eight cores separately, and the total time will be about 100 seconds.

MPjobs is modularized into the DCAT GUI so that it will control multiple runs in parallel, calling the function of MainDCAT. To perform an MPjobs run, in the [Folders] section of the DCAT configuration file, set “PerformMPjobs” = 1.

MPjobs is configured to run DCAT cases with two scenarios, 1D and 2D:

- 1D (idv loop)

A list of .idv files showing different contingency names (termed “**Xfile**” in configuration), e.g., \Examples\PolishSystem_MPjobs\idv\1D.lst, as shown in Figure 6.5.



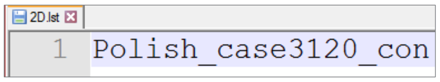
```
initiating_event_PS
initiating_event_PS_1
initiating_event_PS_2
initiating_event_PS_3
initiating_event_PS_4
initiating_event_PS_5
```

Figure 6.5. Contingency .idv file list

The .idv file names (no extension) listed in this file should be saved in the same folder as the 1D list file. MPjobs will conduct the DCAT runs for cases using each of the .idv files. The results for each case will be saved in a separate folder named after the .idv file name. For the 1D case, the user does not need to define “Yfile.”

- **2D (sav/snp loop)**

A list of .idv files showing different contingency names (termed “**Xfile**”) and a list of .sav and .snp files showing different base-case names (termed “**Yfile**”), e.g.,\Examples\ PolishSystem_MPjobs\2D.lst



```
Polish_case3120_con
```

Figure 6.6. Filename list for .sav and .snp files

A pair of .sav and .snp files for the same base-case should share the same filename but different extensions. The .sav/.snp file names (no extension) listed in Yfile should be saved in the same folder as the “2D list” file. Note that the exact file names of .sav and .snp files may not be the same, but both of them must contain the listed string in their file names. MPjobs will recognize the pair of files as a listed case. MPjobs will conduct the DCAT runs by looping over the 2D list first, then looping over the 1D list. The results for each case will be saved in a separate folder named after the combination of .idv file name and sav/snp file name (e.g., initiating_event_PS_1_ Polish_case3120_con). For a 2D case, the user needs to define “Xfile” and “Yfile.”

In our example, there is only one base-case name listed in “2D.lst,” and six contingency names listed in “1D.lst.” MPjobs will perform multicore runs for 1 × 6 = 6 separate cases. Table 6.1 summarizes the differences among a single DCAT run, batch mode, and MPjobs.

Table 6.1. Differences between single run, batch run, and MPjobs run

	Single Run	Batch Run	MPjobs Run
No. CPUs	1	1	Multiple, defined by user
No. Cases	1	Multiple	Multiple, determined by 1D/2D list
No. .idv files	1	One for each case	One for each case
No. INI files	A single file	A common file for all cases OR a specified file for each case	A common file for all cases with an additional [MPjobs] section
No. Output folders	1	One for each case	One for each case

6.5 "MainDCAT.py" (Main DCAT Process Given a Certain Contingency)

"MainDCAT.py" is the main Python module in the DCAT package, and it controls the computational flow of DCAT. It can simulate the potential cascading events of a given contingency. Its core modules are in the module package "RunDCAT" that includes

1. Dynamic Simulation related functions in the "DynamicSimulation" module
2. Special Protection System (SPS) related functions in the "TestSPS" module
3. AC Contingency Analysis related functions in the "TestCorrectiveAction" module
4. After-DCAT related functions in the "AfterDCAT" module
5. DCAT simulation-related functions in the "DCAT_Functions" module.

"MainDCAT" also includes a post-processing module, "ReadLogFile."

6.6 Main Flow of "MainDCAT.py"

The main flow of the Python module "MainDCAT" is illustrated inside the yellow square in Figure 6.7. A filled yellow ellipse represents a module, while filled yellow rectangles represent the functions in the "DCAT_Functions" module.

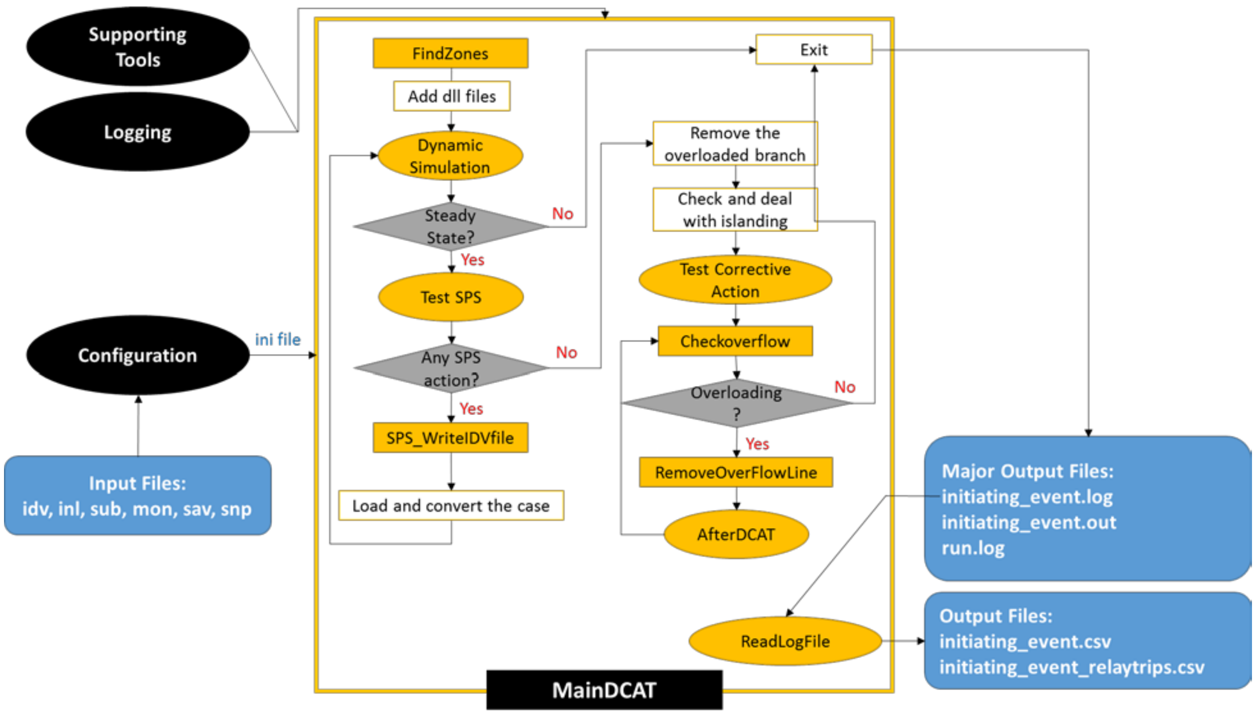

Supporting Tools
Logging
Configuration
ini file
Input Files: idv, inl, sub, mon, sav, snp
FindZones
Add dll files
Dynamic Simulation
Steady State?
No
Yes
Test SPS
Any SPS action?
No
Yes
SPS_WriteIDVfile
Load and convert the case
Exit
Remove the overloaded branch
Check and deal with islanding
Test Corrective Action
Checkoverflow
Overloading ?
No
Yes
RemoveOverFlowLine
AfterDCAT
ReadLogFile
MainDCAT
Major Output Files: initiating_event.log initiating_event.out run.log
Output Files: initiating_event.csv initiating_event_relaytrips.csv

Figure 6.7. Flowchart of DCAT (2019 version) represented by modules

6.7 Functions and Modules Imported into “MainDCAT.py”

6.7.1 “RunDynamicSimulation” in “DynamicSimulation.py”

The “RunDynamicSimulation” function 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
- The control parameters of dynamic simulation
- The channel file defined in configuration file contains settings for channels.

Output(s)

- The number of iterations needed for the dynamic simulation to reach a new quasi-steady state
- The end time (unit: seconds) of the dynamic simulation.

This function can judge whether the dynamic simulation has reached a steady state or not, and if not, another five 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 6.8 ("MaxNumIter" is the "nMax" in Figure 6.8).

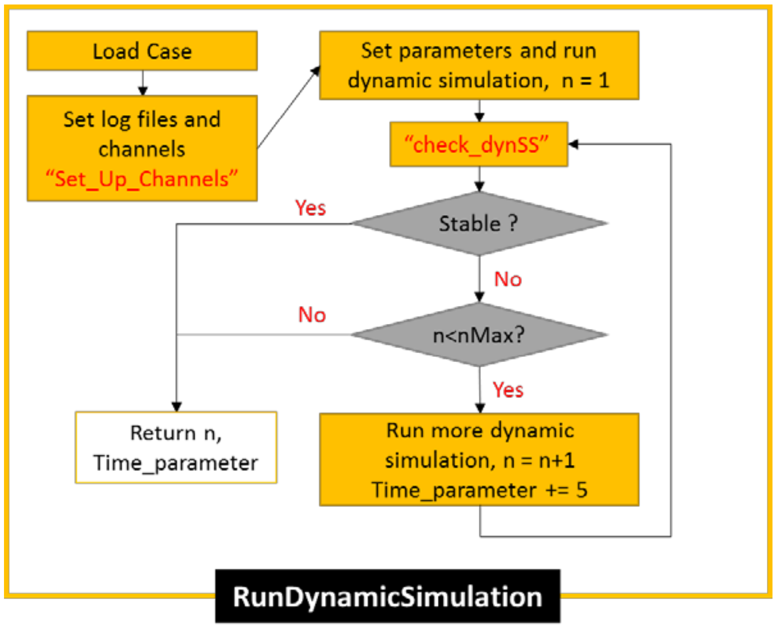

Load Case
Set log files and channels
"Set_Up_Channels"
Set parameters and run dynamic simulation, n = 1
"check_dynSS"
Yes
Stable ?
No
No
n<nMax?
Yes
Return n, Time_parameter
Run more dynamic simulation, n = n+1 Time_parameter += 5
RunDynamicSimulation

Figure 6.8. Flowchart of "RunDynamicSimulation"

6.7.2 "FindZones" in "DCAT_Functions.py"

Th "FindZones" function reads the ".sub" file and returns a list of zone numbers listed in the ".sub" file. If the ".sub" file cannot be found, the function will return a default list of zone numbers that are specified in the input configuration file.

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

6.7.3 "Test_SPS" in "TestSPS.py"

The "Test_SPS" function 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 DCAT
- SetSPSFile: the file defined in the configuration file contains parameters for SPS.

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 the "Test_SPS" subroutine is illustrated in Figure 6.9.

Output(s)

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

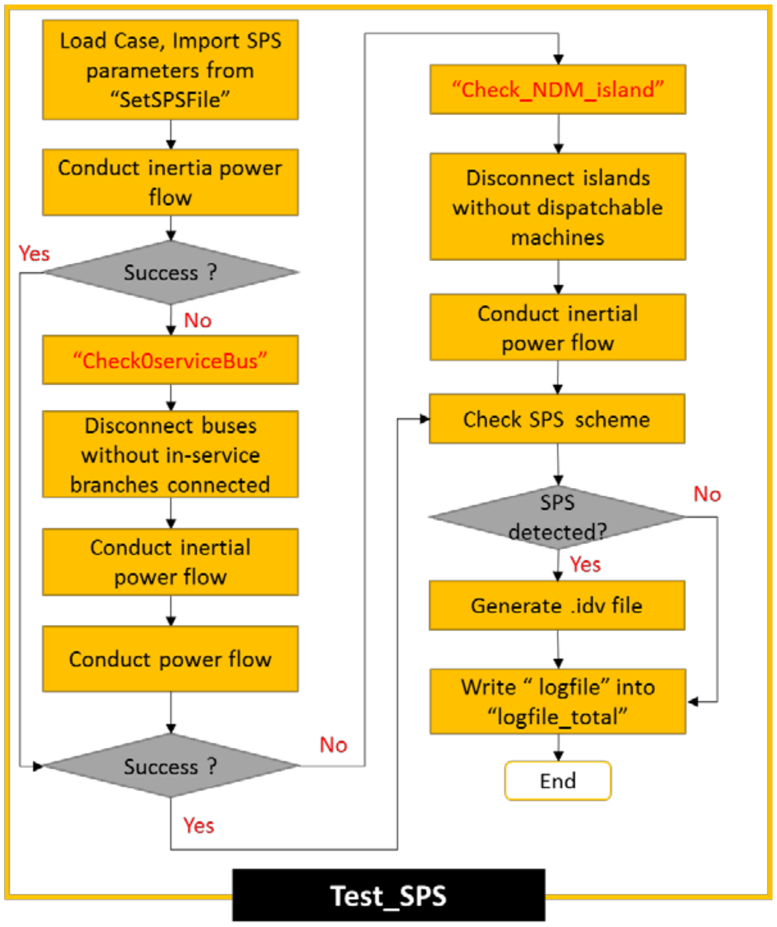

Load Case, Import SPS parameters from "SetSPSFile"
Conduct inertia power flow
Success ?
Yes
No
"CheckOserviceBus"
Disconnect buses without in-service branches connected
Conduct inertial power flow
Conduct power flow
Success ?
No
Yes
"Check_NDM_island"
Disconnect islands without dispatchable machines
Conduct inertial power flow
Check SPS scheme
SPS detected?
No
Yes
Generate .idv file
Write " logfile" into "logfile_total"
End
Test_SPS

Figure 6.9. Illustration of main flow of "Test_SPS"

6.7.4 "SPS_WriteIDVfile" in "DCAT_Functions.py"

The "SPS_WriteIDVfile" function 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.

6.7.5 “Test_CorrectiveAction” in “TestCorrectiveAction.py”

The “Test_CorrectiveAction” function 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
- The name of the “.sub” file
- The name of the “.mon” file.

Output(s):

- A label indicating whether ACCC succeeds or not.

The main flow of this sub-function is presented in Figure 6.10.

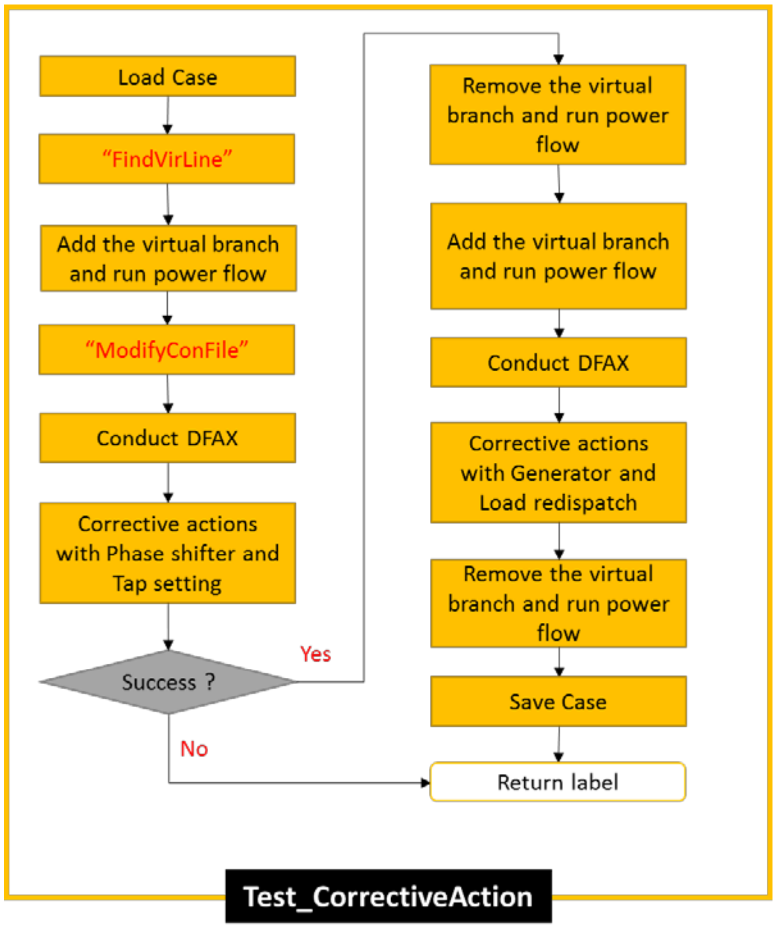

Load Case
"FindVirLine"
Add the virtual branch and run power flow
"ModifyConFile"
Conduct DFAX
Corrective actions with Phase shifter and Tap setting
Success ?
Yes
No
Remove the virtual branch and run power flow
Add the virtual branch and run power flow
Conduct DFAX
Corrective actions with Generator and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

Figure 6.10. Illustration of main flow of "Test_CorrectiveAction." DFAX stands for distribution factor data file.

6.7.6 "Checkoverflow" in "DCAT_Functions.py"

The "Checkoverflow" 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 function "FindZones"
- PER: the threshold to judge overloading, e.g., if PER = 100, the threshold will be 100% of the line rating
- RATE_overloading: which rating of the line will be adopted to judge the overloading: Rate A, Rate B, or Rate C.

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, such as bus numbers, branch id and overloading percentage.

6.7.7 "RemoveOverflowLine" in "DCAT_Functions.py"

The "RemoveOverflowLine" function is called when the function "Checkoverflow" finds and returns the information on the branch with maximum overloading. It produces an ".idv" file to trip the corresponding overloaded 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, such as bus numbers, branch id and overloading percentage
- 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.

6.7.8 "AfterDCAT" in "AfterDCAT.py"

The "AfterDCAT" function 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; or (3) the dynamic simulation cannot initialize properly. The "AfterDCAT" function is called after the first iteration of 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 maximum number of 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 the SPS is checked
- 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 for during the ACCC
- The name of the ".sav" file after the overloaded branch is removed and island checking is complete; (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(bus numbers, branch id, etc.).

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 function is shown in Figure 6.11.

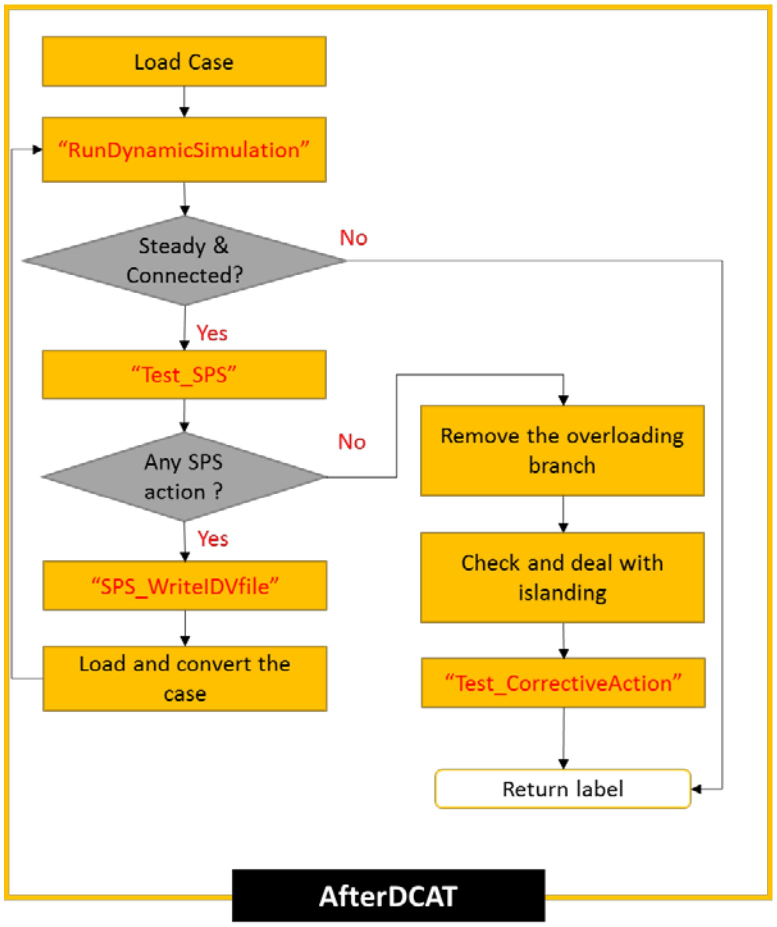


Figure 6.11. Illustration of main flow of "AfterDCAT"

6.7.9 "IsDisconnected" in "DCAT_Functions.py"

The "IsDisconnected" function detects whether all the buses have been disconnected from the system. Disconnection 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.

6.7.10 "FindVirLine" in "TestCorrectiveAction.py"

The "FindVirLine" function is used to find a virtual branch in the open case in order to run the AC contingency analysis. This function basically searches for an existing non-transformer branch and then returns the information of a new branch in parallel with that branch.

Input(s)

- The “FindVirLine” function 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.

6.7.11 “ModifyConFile” in “TestCorrectiveAction.py”

The “ModifyConFile” function 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.

6.8 “GUI.py” (DCAT GUI)

“GUI.py” is the GUI module for the DCAT 2019 version; the user must choose “GUI.py” in the main folder and run the script. A detailed introduction to this module is provided in Section 4.0.

7.0 Data Preparation, Configuration Settings, and Scripting

This section provides generic information on how to prepare data, configuration settings, and scripting for various DCAT runs.

7.1 Preparation of Power System Data

To run the test examples with this DCAT package, the user does not need do any case preparation. If a user wants to use the DCAT 2.0 package to run simulations using other than the test system, the user must do some preprocessing. See Section 7.4 for quick-start instructions on how to set up a DCAT simulation for a different base case. For more advanced users, this section provides generic information on how to convert and modify a base case for a given system using the Python modules provided in the DCAT package.

7.1.1 Modeling of SPS/RAS

One way to model SPS in DCAT is to check system conditions after a system reaches the steady state. If conditions of SPS are met, SPS might implement further system tripping, which will take place as the beginning of the next cascading stage.

The definitions of this type of SPS should be specified by a Python class "SPS.py" for psspy.

7.1.2 Channels

For each system, the channels need to be set up for dynamic simulation. The definitions of this type of information should be specified by a Python class "Channels.py" for the psspy module.

7.2 Details on Configuration Settings

The control interface between the user and DCAT is a configuration file. An example look of the configuration file is shown in Figure 7.1. Each simulation requires a unique configuration file defined by the user. The new configuration file is the .ini file, which is also organized into six separate sections, each of which contains a few setting parameters. The six sections are [Folders], [SystemRelated], [ControlFiles], [StaticInputs], [Parameters], and [MPjobs]:

1. [Folders]: Define the path and the maximum bus size of PSS/E for initialization; define the path of the input folder for the .idv file, the path of the output folder for the .out file and logs, and a subfolder, "Cases," which will store temporary files. The following are optional settings:

   – "PSSEPath" – PSS/E Path on local computer for PSS/E executables. This parameter controls which version of PSS/E DCAT will use. For example, by setting "PSSEPath = C:/Program Files (x86)/PTI/PSSE**34**/PSSBIN," DCAT will execute with the locally installed 34 version.
   "PSSEPath = C:/Program Files (x86)/PTI/PSSE**33**/PSSBIN" will let DCAT run with the locally installed 33 version.

   – "PSSESize" – The maximum number of buses PSS/E allows

   – "InputFolder" – Input folder for the ".idv" file

   – "OutputFolder" – Output folder for the ".out" file and logs; a subfolder, "Cases," will store temporarily files.

   - "System" – For certain systems, additional settings might be required for dynamic simulation. This setting is optional when system-specific codes are added; the default is none.
   - "LIBpath" – Path to the library files (*.dll) needed for specified cases; the default is none, i.e., that no library files are needed.
   - "PerformMPjobs" – If this is set to 1, DCAT performs MPjobs (parallel processing mode). Additional configuration settings are required in the section [MPjobs]; the default is 0.

2. [SystemRelated]: Defines the system-related settings required by PSS/E.
   - "Zone_list" – the default list of the zone number if no information included in ".sub" file

The following 11 parameters are settings for the third release of corrective action function (psspy.accor_3 described in the PSS/E 34 API information[1]) in "RunDCAT\Test_CorrectiveAction.py":

• "RatingSet"	OPTIONS(6) for psspy.accor_3
• "MWmis"	VALUES(1) for psspy.accor_3
• "Volt_tol"	VALUES(3) for psspy.accor_3
• "Branch_tol"	VALUES(4) for psspy.accor_3
• "Gen_ctrlfact"	VALUES(5) for psspy.accor_3
• "Load_ctrlfact"	VALUES(6) for psspy.accor_3
• "PhaseShifter_wt"	VALUES(7) for psspy.accor_3
• "GenOffline_wt"	VALUES(8) for psspy.accor_3
• "TapSetting_wt"	VALUES(9) for psspy.accor_3
• "SwitchedShunt_wt"	VALUES(10) for psspy.accor_3
• "SubsystemLabels"	LABELS(1-6) for psspy.accor_3

The following four parameters are settings for psspy.conl described in the PSS/E API:[1]

• "PconsCurPer"	percentage of active power load to be converted to the constant current characteristic
• "PconsImpPer"	percentage of active power load to be converted to the constant admittance characteristic
• "QconsCurPer"	the percentage of reactive power load to be converted to the constant current characteristic
• "QconsImpPer"	percentage of reactive power load to be converted to the constant admittance characteristic

The following two files are additional inputs for certain systems where a user wants to run simulations with SPS and specific channel information:

- "ChannelFile" – Python class: Set Up Channels for Dynamic Simulation
- "SetSPSFile" – Python class: Set Up SPS-related parameters

[1] …..\Program Files (x86)\PTI\PSSE33\DOCS\API.pdf

3. [ControlFiles]: Define the names for the .idv file (input, specify the contingency) and .out file (output, the major output file, which includes channel returned values).

   The initial contingency file

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

   The name of the “.out” file to be generated

   - “outfile” – This specifies the name of the .out file.

4. [StaticInputs]: Define the names for .sub, .mon, .inl, .sav, and .snp files that store in [Folder].

   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 7.1.1. 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 7.1.2. 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.”

   Files needed for running the inertial 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.
   - “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”.
   - “mon_file”: This specifies the name of the “.mon” (monitoring) file that is going to be adopted for ACCC process.

5. [Parameters]: Define the parameters for dynamic simulation, such as time, time step, tolerance, maximum iteration, etc. This defines the parameters to judge overloading.

   Parameters related to dynamic simulation

   - “Time_parameter”: This determines the time duration of the dynamic simulation.
   - “MaxNumIter”: This is the maximum number of iterations of dynamic simulation that will be conducted.
   - “timestep1”: This is the simulation time step.
   - “filter1”: This is the filter time constant used in calculating bus frequency deviations.

   Parameters related to judging the steady state

   - “TOL”: This determines the tolerance for judging whether the dynamic simulation has reached the steady state.
   - “ck_settle_time”: time interval for comparing with “TOL.” For example, if ck_settle_time = 2, then beyond 10 seconds (Time_parameter) of the dynamic run, the simulation results of the previous two

seconds will be scanned, 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 percentage to judge overloading; e.g., if PER = 100, the threshold will be 100% of the line rating.
- "RATE_overloading": defines which line rating will be adopted to determine overloading: PCTMVARATEA (percent MVA Rate A) stands for Rate A, PCTMVARATEB stands for Rate B, and PCTMVARATEC stands for Rate C. Refer to Samaan et al. (2015) for other options. (Note: Rate C is adopted here. During the case preparation step, the Python module "MakeSAVE.py" modifies the Rate C of the branches to be the lower of 130% of Rate A and 115% of Rate B, based on general practice for longer time allowed overloading).

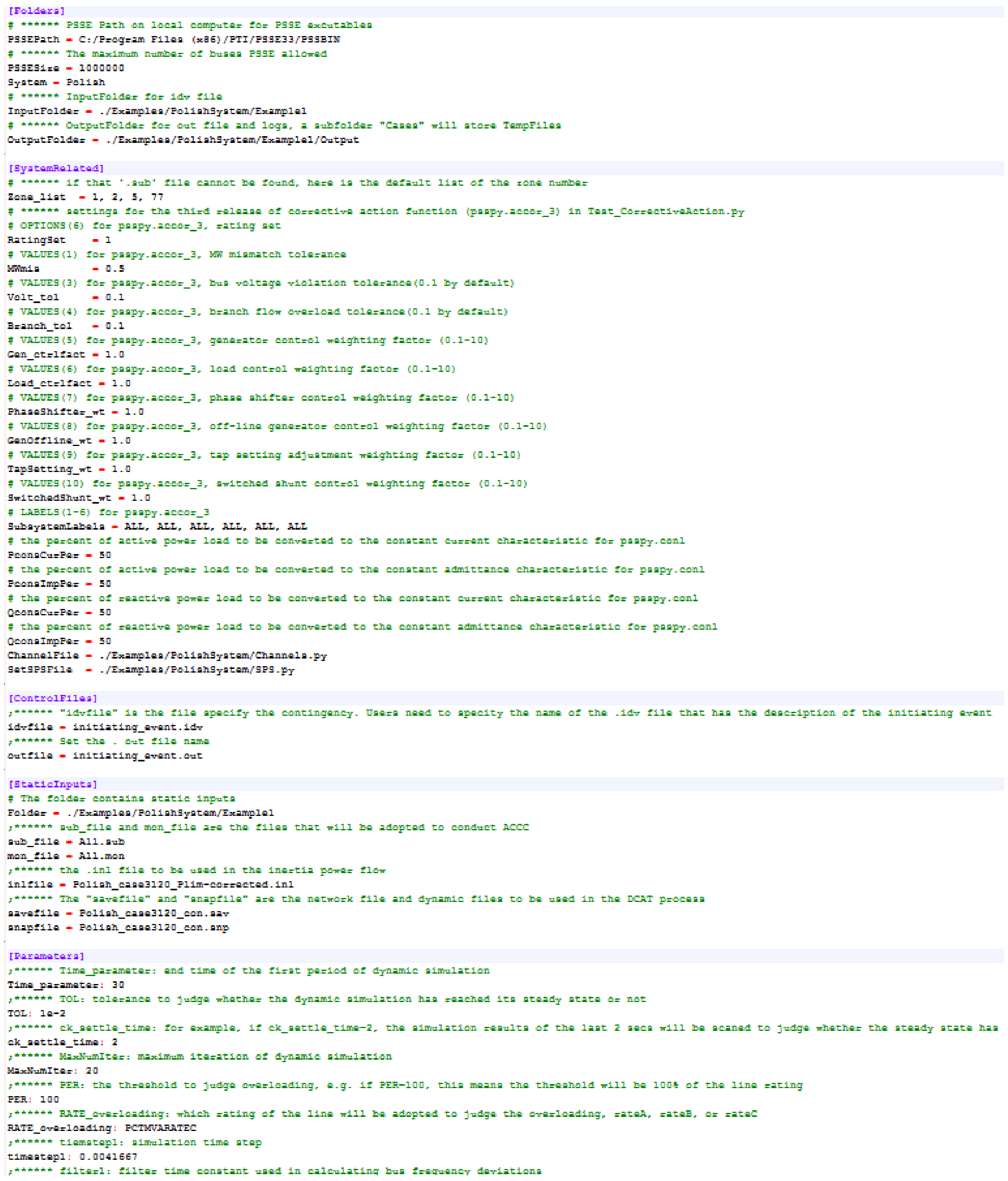
6. [MPjobs]: This section is only available when "PerformMPjobs" = 1 in [Folders].
   - "CPU": an integer; the number of processors to be used (a setting of 0 indicates using all the available processors).
   - "Scenario": "1D" or "2D"; the type of loops
   - "Outspath": path name of a folder saving all the results folders from MPjobs run
   - "Xfile": a file that lists all the *.idv files for 1D and 2D scenarios
   - "Yfile": a file that lists all the *.sav/snp files for 2D scenarios.

Note that the following parameters in the same .ini file are useless for MPjobs and thus will be ignored:

- [Folders]: InputFolder, OutputFolder
- [ControlFiles]: .idvfile
- [StaticInputs]: savefile, snapfile (for 2D scenario)

For an .ini file performing MPjobs, the values for the above parameters can be left blank. All the other configuration settings defined in the .ini file will be the same for each of the cases in a MPjobs run.

Table 7.1 summarized the formats and example values for configuration parameters, corresponding to the definitions in previous paragraphs. Besides the files defined in this configuration file, other files may be generated for different DCAT simulations. The user does not need to define the names, because they will be automatically saved in the output folder. For example, all the log files will be generated automatically according to the name of the .out file and saved in the output folder. The corresponding description for each setting is automatically included when a configuration file is saved through the DCAT GUI.



```
[Folders]
# ****** PSSE Path on local computer for PSSE executables
PSSEPath = C:/Program Files (x86)/PTI/PSSE33/PSSBIN
# ****** The maximum number of buses PSSE allowed
PSSESize = 1000000
System = Polish
# ****** InputFolder for idv file
InputFolder = ./Examples/PolishSystem/Example1
# ****** OutputFolder for out file and logs, a subfolder "Cases" will store TempFiles
OutputFolder = ./Examples/PolishSystem/Example1/Output

[SystemRelated]
# ****** if that '.sub' file cannot be found, here is the default list of the zone number
Zone_list  = 1, 2, 5, 77
# ****** settings for the third release of corrective action function (psspy.accor_3) in Test_CorrectiveAction.py
# OPTIONS(6) for psspy.accor_3, rating set
RatingSet     = 1
# VALUES(1) for psspy.accor_3, MW mismatch tolerance
MWmis         = 0.5
# VALUES(3) for psspy.accor_3, bus voltage violation tolerance(0.1 by default)
Volt_tol      = 0.1
# VALUES(4) for psspy.accor_3, branch flow overload tolerance(0.1 by default)
Branch_tol    = 0.1
# VALUES(5) for psspy.accor_3, generator control weighting factor (0.1-10)
Gen_ctrlfact = 1.0
# VALUES(6) for psspy.accor_3, load control weighting factor (0.1-10)
Load_ctrlfact = 1.0
# VALUES(7) for psspy.accor_3, phase shifter control weighting factor (0.1-10)
PhaseShifter_wt = 1.0
# VALUES(8) for psspy.accor_3, off-line generator control weighting factor (0.1-10)
GenOffline_wt = 1.0
# VALUES(9) for psspy.accor_3, tap setting adjustment weighting factor (0.1-10)
TapSetting_wt = 1.0
# VALUES(10) for psspy.accor_3, switched shunt control weighting factor (0.1-10)
SwitchedShunt_wt = 1.0
# LABELS(1-6) for psspy.accor_3
SubsystemLabels = ALL, ALL, ALL, ALL, ALL, ALL
# the percent of active power load to be converted to the constant current characteristic for psspy.conl
PconsCurPer = 50
# the percent of active power load to be converted to the constant admittance characteristic for psspy.conl
PconsImpPer = 50
# the percent of reactive power load to be converted to the constant current characteristic for psspy.conl
QconsCurPer = 50
# the percent of reactive power load to be converted to the constant admittance characteristic for psspy.conl
QconsImpPer = 50
ChannelFile = ./Examples/PolishSystem/Channels.py
SetSPSFile  = ./Examples/PolishSystem/SPS.py

[ControlFiles]
;****** "idvfile" is the file specify the contingency. Users need to specity the name of the .idv file that has the description of the initiating event
idvfile = initiating_event.idv
;****** Set the . out file name
outfile = initiating_event.out

[StaticInputs]
# The folder contains static inputs
Folder = ./Examples/PolishSystem/Example1
;****** sub_file and mon_file are the files that will be adopted to conduct ACCC
sub_file = All.sub
mon_file = All.mon
;****** the .inl file to be used in the inertia power flow
inlfile = Polish_case3120_Plim-corrected.inl
;****** The "savefile" and "snapfile" are the network file and dynamic files to be used in the DCAT process
savefile = Polish_case3120_con.sav
snapfile = Polish_case3120_con.snp

[Parameters]
;****** Time_parameter: end time of the first period of dynamic simulation
Time_parameter: 30
;****** TOL: tolerance to judge whether the dynamic simulation has reached its steady state or not
TOL: 1e-2
;****** ck_settle_time: for example, if ck_settle_time=2, the simulation results of the last 2 secs will be scaned to judge whether the steady state has
ck_settle_time: 2
;****** MaxNumIter: maximum iteration of dynamic simulation
MaxNumIter: 20
;****** PER: the threshold to judge overloading, e.g. if PER=100, this means the threshold will be 100% of the line rating
PER: 100
;****** RATE_overloading: which rating of the line will be adopted to judge the overloading, rateA, rateB, or rateC
RATE_overloading: PCTMVARATEC
;****** tiemstep1: simulation time step
timestep1: 0.0041667
;****** filter1: filter time constant used in calculating bus frequency deviations
```

Figure 7.1. New configuration file (.ini file) of DCAT (2019 version)

Table 7.1. Setting parameters defined in configuration file

Section	Parameter Name	Format	Default/Example	Included in Folder
Folder	PSSEPath	String (Path Name)	C:/Program Files (x86)/PTI/PSSE33/PSSBIN	
	PSSESize	Integer	1000000	
	InputFolder	String (Path Name)	./Examples/Example1*	
	OutputFolder	String (Path Name)	./Examples/Example*/Results	
	LIBpath (Optional)	String (Path Name, default is None)	./Input/LIBs	
	System (Optional)	String	SAVNW	
	PerformMPjobs (Optional)	0 (default) or 1	0	
SystemRelated (Parameters for psspy functions)	Zone_list	Integer List (separated by comma)	1, 2, 5, 77	
	RatingSet	Decimal	3	
	Mwmis	Decimal	1.5	
	Volt_tol	Decimal	0.1	
	Branch_tol	Decimal	0.1	
	Gen_ctrlfact	Decimal	1	
	Load_ctrlfact	Decimal	1	
	PhaseShifter_wt	Decimal	1	
	GenOffline_wt	Decimal	1	
	TapSetting_wt	Decimal	1	
	SwitchedShunt_wt	Decimal	1	
	PconsCurPer	Decimal, percentage	50	
	PconsImpPer	Decimal, percentage	50	
	QconsCurPer	Decimal, percentage	50	
	QconsImpPer	Decimal, percentage	50	
	ChannelFile	File name (*.py)	./Examples/Example1/ Channels.py	
	SetSPSFile	File name (*.py)	./Examples/Example1/ SPS.py	
ControlFiles	idvfile	*.idv	initiating_event_XX.idv	InputFolder
	outfile	*.out	initiating_event.out	OutputFolder

Section	Parameter Name	Format	Default/Example	Included in Folder
StaticInputs	folder	String (Path Name)	./Input	
	sub_file	*.sub	Allsub	Folder
	mon_file	*.mon	All.mon	Folder
	inlfile	*.inl	savnw_Plim-corrected.inl	Folder
	savefile	*.sav	savnw_con.sav	Folder
	snapfile	*.snp	savnw_con.snp	Folder
Parameters	time_parameter	Integer, unit: second	30	
	tol	Decimal, unit: pu	0.01	
	ck_settle_time	Integer, unit: second	2	
	maxnumiter	Integer	20	
	per	Integer, unit: percentage (%)	100	
	rate_overloading	String	PCTMVARATEC	
	timestep1	Decimal, unit: second	0.0041667	
	filter1	Decimal, unit: second	0.0166668	
MPjobs (only included when PerformMPjobs = 1)	CPU	Integer	8	
	Scenario	String, “1D” or “2D”	2D	
	Outspath	String (Path Name)	./Examples/Example1	
	Xfile	File Name for 1D and 2D	./Examples/ Example1/1D.lst	
	Yfile	File Name for 2D	./Examples/ Example1/2D.lst	
(Not included in Configuration File; Files are created using default names)	spsfile	*.idv	SPS_action1.idv	<outputfolder>/ Cases
	spsfile_after	*.idv	SPS_action1_aftercontingency .idv	<outputfolder>/ Cases
	idvfile_react_sps	*.idv	idvfile_react_SPS.idv	<outputfolder>/ Cases
	idvfile_react_sps_af ter	*.idv	idvfile_react_SPS_aftercontin gency.idv	<outputfolder>/ Cases
	confile	*.con	temp.con	<outputfolder>/ Cases
	dfxfile	*.dfx	test_corrective_actions.dfx	<outputfolder>/ Cases
	dfxfile_after	*.dfx	test_corrective_actions.dfx	<outputfolder>/ Cases

Section	Parameter Name	Format	Default/Example	Included in Folder
	idvfile_reoverflow	*.idv	idvfile_reoverflow.idv	<outputfolder>/ Cases
	returnfile_sps	*.sav	AfterDynamicSim_sps.sav	<outputfolder>/ Cases
	returnfile_sps_after	*.sav	AfterDynamicSim_sps_aftercontingency.sav	<outputfolder>/ Cases
	returnfile	*.raw	After_dynamic_raw_file.raw	<outputfolder>/ Cases
	returnfile_sav	*.sav	AfterDynamicSim_corr.sav	<outputfolder>/ Cases
	returnfile_raw	*.raw	AfterDynamicSim_corr.raw	<outputfolder>/ Cases
	savefile_after	*.sav	AfterDynamicSim_corr_convert.sav	<outputfolder>/ Cases
	returnfile_after	*.raw	After_dynamic_raw_file_after contingency.raw	<outputfolder>/ Cases
	casefile_after	*.sav	AfterDynamicSim_sps_aftercontingency_convert.sav	<outputfolder>/ Cases
	returnfile_sav_after	*.sav	AfterDynamicSim_corr_aftercontingency.sav	<outputfolder>/ Cases
	returnfile_raw_after	*.raw	AfterDynamicSim_corr_aftercontingency.raw	<outputfolder>/ Cases
	logfile	*.log	<outfile>.log	outputfolder
	logfile2	*.log	<outfile>_promptoutput.log	outputfolder
	logfile_corr	*.log	<outfile>.log	outputfolder
	logfile_after	*.log	<outfile>.log	outputfolder
	logfile2_after	*.log	<outfile>_promptoutput.log	outputfolder
	logfile_corr_after	*.log	<outfile>.log	outputfolder
	SPS_logfile	*.log	test_SPS.log	outputfolder
	SPS_logfile_after	*.log	test_SPS.log	outputfolder

7.3 Alternative Method for Testing DCAT Simulation without GUI

The easy way to execute DCAT is to use the GUI (Section 4); the other method is to call the main function in “MainDCAT.py” after the configuration file is loaded. An example script (“Test.py”) is included in the DCAT package.

The corresponding program settings are located in the “Config.ini” file, in the example subfolders, e.g., “Examples\PolishSystem_MPjobs”. After modifying the configuration parameters (Section 7) 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 6 describes some further modifications that users might need for real cases.

7.4 Quick-Start Instructions for using DCAT on Another Base Case

This section provides instructions on how a user can quickly set up a DCAT simulation on another prepared PSS/E base case, without altering the Python modules provided in the DCAT package. The user will need to have the DCAT package from Section 4.0 installed.

Quick DCAT Set-Up Instructions

A. For MPjobs mode

1. Create a new folder.
2. Ensure the power flow solves in PSS/E for the base cases you wish to use.
3. Prepare the .sav and .snp files for the solved power flow base cases in Step 2.
4. Prepare the .idv files with the desired contingency definitions.
5. Prepare the “1D.lst” file with the names of the .idv files created in Step 4.
6. Prepare the “2D.lst” file with the file names of the .sav/.snp files created in Step 3.
7. Collect a copy of the “Channels.py” from Examples. You do not need to change this file.
8. Collect a copy of the “SPS.py” file from Examples. You only need to change this file if you want to define SPS.
9. Put all the files prepared in Steps 3–8 in the new folder created in Step 1.
10. Prepare the “Configuration Settings” file.
    - Collect a copy of the “Configuration Settings” file from “PolishSystem_MPjobs” Examples.
    - Set the parameters ChannelFile, SetSPSFile, Outspath, Xfile, and Yfile to point to the name of the new folder you created in Step 1.
11. Put the modified “Configuration Settings” file in the new folder created in Step 1.
12. Run the GUI.py script and choose File → Open, and select the “Configuration Settings” file you created in Step 10.
13. Click “Run MPjobs” and wait for the DCAT simulations to complete.
14. Find the results of your simulation in the new folder you created in Step 1.

B. For a single run

1. Create a new folder.
2. Ensure the power flow solves in PSS/E for the base case you wish to use.
3. Prepare the .sav and .snp files for the solved power flow base case in Step 2.
4. Prepare the .idv file with the desired contingency definition.
5. Collect a copy of the “Channels.py” from Examples. You do not need to change this file.

6. Collect a copy of the “SPS.py” file from Examples. You only need to change this file if you want to define SPS.
7. Put all the files prepared in Steps 3–8 in the new folder created in Step 1.
8. Prepare the “Configuration Settings” file:
   – Collect a copy of the “Configuration Settings” file from “PolishSystem” or “Test” Examples.
   – Set the parameters ChannelFile, SetSPSFile, InputFolder, OutputFolder, .idvfile, savefile, and snapfile to point to the name of the new folder you created in Step 1.
9. Put the modified “Configuration Settings” file in the new folder created in Step 1.
10. Run the GUI.py script and choose File → Open and select the “Configuration Settings” file you created in Step 8.
11. Click “Run DCAT” and wait for the DCAT simulations to complete.
12. Find the results of your simulation in the new folder you created in Step 1.

7.5 Scripting

The following scripts are examples to show DCAT users how to use the modules in a script for different simulation goals.

A. Run a single DCAT simulation by importing the configuration file using the “Configuration” module:

```
# Import the required DCAT modules: Configuration and MainDCAT
from DataReader.Configuration import ConfigSettings
from MainDCAT import Main

# Initialize the class of ConfigSettings
config = ConfigSettings()
# Read in the configuration file
config.GetFromINI("./Examples/Test/Example1/Config.ini")
# Examine the existence of the input files
config.ExamineInputFiles()
# Call Main DCAT to perform the simulation
Main(config)
```

B. Run batch processing by importing the “BatchProcessing” module:

```
# Import BatchProcessing Module
from Utils.BatchProcessing import BatchProcessing
# Perform Batch Mode
BatchProcessing("./Examples/Test")
```

C. Run parallel processing by importing the “MPjobs” module:

```
from DataReader.Configuration import ConfigSettings
from MPjobs import MPjobs_Main

if __name__ == '__main__':
 # Initialize the class of ConfigSettings
 config = ConfigSettings()
 # Read in the configuration file
```

```
config.GetFromINI("./Examples/PolishSystem_MPjobs/Config_33.ini")
# Perform MPjobs
MPjobs_Main(config)
```

8.0 References

Samaan NA, JE Dagle, YV Makarov, R Diao, MR Vallem, TB Nguyen, LE Miller, BG Vyakaranam, S Wang, FK Tuffner, and MA Pai. 2015. "Dynamic Contingency Analysis Tool – Phase 1." PNNL-24843, Pacific Northwest National Laboratory, Richland, Washington. Available at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-4843.pdf.

Conto J. 2015. "Contingency Analysis with MPjobs." *Power System topics*. Accessed July 30, 2019, at http://joseconto.blogspot.com/2015/06/.

Appendix A – System Requirement Details

This appendix provides more details on system requirements for performing DCAT studies.

A.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."

A.2 Steps to Create a Python Project

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

A.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.
- Choose "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.