University of Southern Queensland

Faculty of Engineering & Surveying

SOFTWARE DEVELOPMENT FOR POWER SYSTEM ANALYSIS

A dissertation submitted by

ONG Gimhua

in fulfillment of the requirements of

Courses ENG 4111 and ENG4112 Research Project

towards the degree of

Bachelor of Electrical and Electronic Engineering

Submitted: October, 2005

Abstract

Power-flow calculation under both normal and abnormal conditions is very important for power system planning and operation. The main objective of this project is to develop a software for power flow calculation.

The software for the power flow calculation has been developed in this project. The software consists of a tool box for drawing the schematic diagram of a power system, a schematic check program to detect the system connection, a database to store system and component data, a simulator program to simulate power flow of power system and animation program to show load flow direction.

A test system has been analysed to test the software. Both normal and abnormal conditions have been successfully simulated.

University of Southern Queensland
Faculty of Engineering and Surveying

ENG4111/2 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof G Baker
Dean
Faculty of Engineering and Surveying

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indication and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

ONG Gimhua

Student number: 0031132222

_____
Signature

_____
Date

Acknowledgments

For the past seven months, while working on this Research Project has indeed been an enriching experience for me. I would like to take this opportunity to express thanks and appreciation to all those who have helped and contributed to the successful completion of this project.

First of all, I would like to thank my supervisor, Mr. Ron Sharma for his valuable advice and guidance throughout the whole duration of the project. He has fostered my independence while at the same time remaining approachable and encouraging. He has indeed played a crucial role in the learning process.

Last but not least, if I have left out anyone, thanks to you. It would not have been possible to complete this project so smoothly without help from all of you.

ONG Gimhua

University of Southern Queensland
October 2005

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The basic function of an electric power system is to supply electrical energy as economically as possible and with acceptable degree of reliability and quality. Electrical power is like the air you breathe: You wouldn't really think about it until it was missing. Power is just "there" meeting your every need, constantly. It is only during a power failure, that you realize how important power is in your daily life. Without it, daily routine life will standstill. Therefore, poor supply reliability in any part of the power system can result in interruption ranging from inconvenience to major and widespread catastrophic disruption of the supply.

The study of dynamic response to sudden changes and disturbances in electrical power systems is carried out in this project. Common disturbances include fault, power flow, switching, load changes, motor starting, loss of generation, loss of excitation, and blocked governor, etc.

A fault in a circuit is any failure which interferes with the normal flow of current. Most faults on transmission lines of 115 kV and higher are caused by lightning, which results in the flashover of insulators. Faults can be very destructive to power system. A great deal of study, development of devices, and design of protection schemes have resulted in continual improvement in the prevention of damage to transmission lines and equipment and interruptions in generation following the occurrence of a fault.

Power flow study is the basic requirement in power system planning, operation, economic scheduling and exchange of power between utilities. A power system usually consists of hundreds of nodes and transmission lines connected into a complicated configuration designated grid. For the normal operation of a power system, the total real and reactive power supply must equal the total real and reactive demand at any time and any system operating conditions. Power flow analysis is used to examine possible network violations (such as voltage, line flow,

stability violations) under different grid operating conditions such as peak load, light load etc.

Electrical engineers are concerned with every step in the process of generation, transmission, distribution and utilization of electrical energy. The electric utility industry is probably the largest and most complex industry in the world. The electrical engineer who works in this industry will encounter challenging problems in designing future power systems to deliver increasing amounts of electrical energy in a safe, clean, and economical manner.

As existing software by *Power System Analysis*, c1999, Hadi Saadat which is in MATLAB code and that make it non-portable. Therefore a windows-based software tool would be developed to conduct performance evaluation of power systems.

1.1 Power System Simulation Software

It enhances the flexibility of design and analysis of electrical power systems. The powerful drawing tools quickly create a structured, interactive one-line diagram system model which allows move, delete, and place components on the screen with the click of the mouse. The software also provides a schematic program to check and calculate load flow and allows existing documents to be saved in the database.

1.2 Objectives

The basic objective of the project is to:

1. Create Tool box for drawing.

2. Create a Schematic check program to check all the interconnections of a busbar with other components are performed to form the NxN system.

3. Design a simulation program to simulate power flow.

4. Draw Animation to show load flow direction.

5. Create a database to store component data and system.

6. Perform DC/AC load analysis.

7. The software intended to use for the above includes Microsoft Visual Basic 6 programming language and Microsoft Access 2000.

1.3 Overview of Dissertation

This dissertation is divided into seven chapters.

Chapter 2 of the dissertation describes the basic concepts of general power system. The technique used to evaluate the load flow like Gauss Seidel is discussed.

Chapter 3 discuss on the basic knowledge of Visual basic.

Chapter 4 explains the software development for Power System Analysis on the DC analysis. It explains how the toolbox, node-to-node detection, animation and load flow simulation are created.

Chapter 5 explains the software development for Power System Analysis on the AC analysis. It explains how AC analysis is being programmed in.

Chapter 6 will show the result of the software being evaluated based on RBTS drawn on both AC and DC analysis.

Chapter 7 covers on the conclusion and recommendations to enhance the software.

Chapter 8 is the references.

Chapter 2

Power Flow Analysis

This chapter gives detailed information on the Power Flow Analysis. It will cover the basic concepts and established terminology. In particular, the following are reviewed: power flow solution by Gauss-Seidel and Newton-Raphson methods.

2.1 Overview

In general, a power system is very complicated physically, and power flow problems can not be solved as a linear system. Its solution is not easy to solve mathematically, and is only possible by numerical iterations. Although there number of methods to solve power flow problems, Gauss-Seidel and Newton-Raphson method will be presented in this chapter.

2.2 Basic Equipment

Although power systems are designed and operated in a three-phase form, because of the important feature of the three-phase system – symmetry, a per phase analysis is usually performed. There are few types of basic equipment in power systems:

Load: Loads, including active and reactive powers, are connected to buses and are known. However, their voltage magnitudes and phase angles are unknown.

Generator: Generators are connected to buses and the active power generated and voltage magnitude are predetermined at each bus. If the reactive power required for holding the specified voltage is within its limits, such a voltage will be maintained. Otherwise, the reactive

power generation will be set at the limit and the voltage magnitude becomes a variable.

Transmission lines:

The material medium or structure that forms all or part of a path from one place to another for directing the transmission of energy, such as electric currents, or magnetic fields.

Circuit Breakers:

Circuit breakers are a center unit used to monitor the amount of current flow and were developed to protect a significant length of wire from damage due to over current conditions.

2.3 Load Flow Studies

Load flow studies in planning the future expansion of power system in determining the nest operation of existing systems. The principal information obtained from a load flow study is provided by the printout of the solution from computer programs used by the power companies. Most of the features will be brought out in the program. There are 2 methods which I came upon, which are Gauss-Seidel and Newton-Raphson. In this project, Gauss-Seidel is used.

2.4 Data Preparation for Load Flow Studies

Mathematically, the network equations can be formed in the bus (or nodal frame of reference, in the loop (or mesh) frame or reference, or in the branch frame of reference. The bus frame of reference is important. The equations may be represented using either impedance or admittance parameters.

In the bus frame of reference, the performance is described by n-1 linear independent equation for n number of nodes. The reference node, which is at ground potential, is always neglected. In the admittance form, the performance equation can be written as

$$\overline{I_B} = \overline{Y_B}\,\overline{V_B}$$

Where $\overline{I_B}$ the vector of injection bus currents is, $\overline{V_B}$ is the vector of bus or nodal voltages measured from the reference node, and $\overline{Y_B}$ is the bus admittance matrix. Making the above equation into matrix form:

$$\begin{vmatrix} I_1 \\ I_2 \\ . \\ I_{(n-1)} \end{vmatrix} = \begin{vmatrix} Y_{11} & Y_{12} & \dots & Y_{1,n-1} \\ Y_{21} & Y_{22} & \dots & Y_{2,n-1} \\ . & . & \dots & . \\ Y_{(n-1),1} & Y_{(n-1),2} & \dots & Y_{(n-1),(n-1)} \end{vmatrix} \begin{vmatrix} V_1 \\ V_2 \\ . \\ V_{n-1} \end{vmatrix}$$

$\overline{Y_B}$ is a nonsingular square matrix of order (n-1)(n-1). It has an inverse:

$\overline{Y_B}$-1 = $\overline{Z_B}$

Where $\overline{Z_B}$ is the bus impedance matrix. With this the equation can be formed as:

$$\overline{V_B} = \overline{Z_B}\,\overline{I_B}$$

For a general network with n + 1 node:

$$\overline{Y} = \begin{vmatrix} Y_{11} & Y_{12} & \cdots & Y_{1n} \\ Y_{21} & Y_{22} & \cdots & Y_{2n} \\ . & . & \cdots & . \\ Y_{n1} & Y_{n2} & \cdots & Y_{nn} \end{vmatrix}$$

Where each admittance Yii (i = 1,2,3,4, …) is the self-admittance or driving point admittance of node i, given by the diagonal elements, and it is equal to an algebraic sum of all admittances terminating in that node. Yik (i,k = 1,2,3,4,….) is the mutual admittance between nodes i and k or transfer admittance between nodes I and k and is equal to the negative of the sum of all admittances directly connected between those nodes. The current entering a node is given by

$$\text{Ik} = \sum_{n=1}^{n} Y_{kn} V_n$$

Unlike the bus admittance matrix, the bus impedance matrix cannot be formed by simple examination of the network circuit. The bus impedance matrix can be formed by the following methods:

- Inversion of the admittance matrix
- By open circuit testing
- By step-by-step formation

Direct inversion of the Y matrix will need certain assumptions in forming the bus impedance matrix:

The passive network can be show within a closed perimeter. It includes the impedances of all the circuit components, transmission lines, loads, transformers, cables, and generators. The network is passive in the sense that no circulating currents flow in the network. Also the load currents are negligible with respect to the fault currents. All terminals marked 0 are at the same potential. All generators have the same voltage magnitude and phase angle and are replaced by one equivalent generator connected between 0 and a node.

Other than the bus admittance matrix, the program needs the Bus Input Data and the Line Input Data.

Bus Input Data:

The format for the bus entry is chosen to facilitate the required data for each bus in a single row. It is in matrix format. Column 1 is the bus number. Column 2 contains the bus code. Column 3 and 4 are voltage magnitude in per unit and phase angles in degrees. Columns 5 and 6 are load MW and Mvar. Column 7 through 10 is MW, Mvar, minimum Mvar and maximum Mvar of generator. The last column is the injected Mvar capacitors.

Line Input Data:

It is identified by the node-pair method. The information required must be included in a matrix. Column 1 and 2 are the line bus numbers, to and fro respectively. Column 3 through 5 contains the line resistance, reactance, and one-half of the total line charging susceptance in per unit. The last column is for the transformer tap setting.

A better illustration on data needed for load flow, Y matrix, Bus Input Data and Line Input Data are given in Figure 2-1:

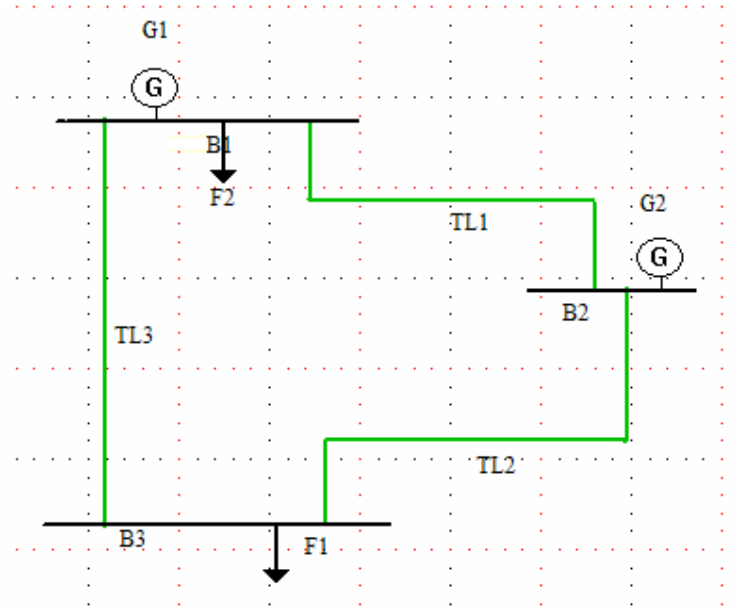


Figure 2-1 Example to show value of Y matrix

Line Impedance/Shunt Admittance:

Z12 = 0.02 + j 0.04

Y12 = 10 – j 20

Z13 = 0.01 + j 0.03

Y13 = 10 – j 30

Z23 = 0.0125 + j 0.025

Y23 = 16 – j 23

Obtain Ybus matrix:

$$\text{Ybus} = \begin{bmatrix} Y_{12}+Y_{13} & -Y_{12} & -Y_{13} \\ -Y_{12} & Y_{12}+Y_{23} & -Y_{23} \\ -Y_{13} & -Y_{23} & Y_{13}+Y_{23} \end{bmatrix}$$

$$\text{Ybus} = \begin{bmatrix} 20-j50 & -10+j20 & -10+j30 \\ -10+j20 & 26-j52 & -16+j32 \\ -10+j30 & -16+j32 & 26-j62 \end{bmatrix}$$

Classify the buses as follows:

Table 2-1 Type of Bus

Bus Type	Given Parameters	Unknown Parameters
Slack Bus	V, δ	P, Q
Generator Bus	P, |V|	Q, δ
Load Bus	P, Q	V, δ

Table 2-2 Variables of Bus

Bus Number	Type	Given	Unknown	Assumption
1	Slack	V1, δ1	P1, Q1	-
2	Load	P2, Q2	|V2|, δ2	|V2| = 1, δ2 =0
3	Voltage	P3, |V3|	Q3, δ3	δ3 = 0

Where:

Slack Bus: the only necessary information is the voltage magnitude and its phase angle.

Load Bus: the loads are entered positive in megawatts and megavars. For this bus, initial voltage estimate must be specified. This is usually 1 and 0 for voltage magnitude and phase angle, respectively. If voltage magnitude and phase angle for this type of bus are specified, they will be taken as the initial starting voltage for that bus instead of a flat start of 1 and 0.

Voltage-Controlled Bus: voltage magnitude, real power generation in megawatts, and the minimum and maximum limits of the magavar demand must be specified.

1. At the slack bus we know both magnitude and phase angle of V so we don't use the formula. But we don't know S, so we calculate it as Vi[summation(VjYij)]*
2. At voltage-controlled bus we know both magnitudes of V but not angle. We know P but not Q, sp we keep the angle calculate from iteration; set V magnitude to known value. We calculate S = Vi[Summation(VjYij)]* = P + jQ; then set P to known value.

2.5 Techniques in Solving Load Flow – Gauss Seidel Method

In load flow calculations the system equations can be written in terms of current, voltage, or power at the *k*th node. We know that the matrix equation in terms of unknown voltages, using the bus admittance matrix for n + 1 nodes is

$$\begin{vmatrix} I_1 \\ I_2 \\ . \\ I_{(n-1)} \end{vmatrix} = \begin{vmatrix} Y_{11} & Y_{12} & \dots & Y_{1,n-1} \\ Y_{21} & Y_{22} & \dots & Y_{2,n-1} \\ . & . & \dots & . \\ Y_{(n-1),1} & Y_{(n-1),2} & \dots & Y_{(n-1),(n-1)} \end{vmatrix} \begin{vmatrix} V_1 \\ V_2 \\ . \\ V_{n-1} \end{vmatrix}$$

Although the currents entering the nodes from generators and loads are not known, these can be written in terms of *P*, *Q*, and *V*:

$$Ik = \frac{P_k - jQ_k}{V_k *}$$

Currents entering the nodes are considered positive, and the power into the node is also positive. A load draws power out of the node and this the active power and inductive vars are entered as: $-P - j(-Q) = -P + jQ$. The current is then $\frac{-P + jQ}{V *}$.

The nodal equal of current at the *k*th node becomes:

$$\frac{P_k - jQ_k}{V_k *} = Yk1V1 + Yk2V2 + Vk3Y3 + \dots + YkkVk + \dots + YknVn$$

In general, for the *k*th node:

$$Vk = \frac{1}{Ykk} \left[\frac{Pk - jQk}{Vk *} - \sum_{i=1}^{i-n} YkiVi \right] + Yi,i\ Vi \text{ for } i \neq k$$

In fact the Gauss-Seidel procedure can be summarized for PQ buses in the following steps:

1. Initial phasor values of load voltages are assumed, the voltage of the generator can be specified. A flat voltage start assumes to be 1 + j0 voltages at buses if not specified.
2. Based on the initial voltages, the voltage at a bus in the first iteration is calculated using the equation above for bus 2.
3. The estimate of the voltage at bus 2 is refined by repeatedly finding new values of *V2* by substituting the value of *V2* into the right-hand side of the equation.
4. The voltages at bus 3 are calculated using the latest value of *V2* found in step 3 and similarly for other buses in the system.

Thus, this would complete one iteration. The iteration process is repeated for the entire network till the specified convergence is obtained.

From the same example above, the iterative method is as follow:

Initial Value:

$$V = \begin{bmatrix} 1.05 \\ 1 \\ 1 \end{bmatrix}$$

Using the iterative formula:

$$V^{(1)} = \begin{bmatrix} 1.05 + j0 \\ 0.969 - j0.0558 \\ 0.99 - j0.0881 \end{bmatrix} \qquad V^{(2)} = \begin{bmatrix} 1.05 + j0 \\ 0.978 - j0.106 \\ 1.007 - j0.063 \end{bmatrix}$$

$$V^{(3)} = \begin{bmatrix} 1.05 + j0 \\ 0.988 - j3.401 \\ 0.9997 - j0.01355 \end{bmatrix}$$

The final solution after 100 iterations:

|V1| = 1.05 arg(V1) = 0

|V2| = 0.982 arg(V2) = -3.502

|V3| = 1.001 arg(V3) = -2.862

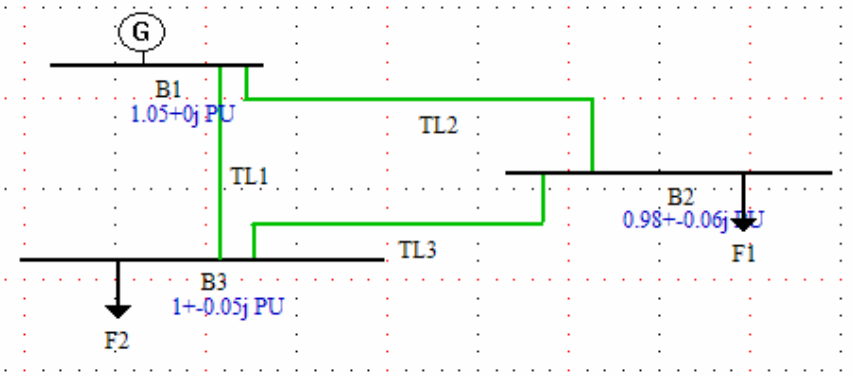


Figure 2-2 Result after gaining voltages (DC)

2.6 Line Flows and Losses

After the iterative solution of bus voltage, the next step is the computation of line flows and line losses.

Given a line connecting two buses *i* and *j*, the line current:

$$Iij = Iℓ + Ii0 = yij\ (Vi - Vj) + yi0Vi$$
$$Iji = -Iℓ + Ij0 = yji\ (Vj - Vi) + yj0Vj$$

The complex powers *Sij* from bus *i* to *j* and *Sji* from bus *j* to *i* are:

$$Sij = Vij\ I^*ij$$
$$Sji = Vji\ I^*ji$$

The power loss in line *i* – *j* is the algebraic sum of the power flows, *Sij* and *Sji*

$$SLij = Sij + Sji$$

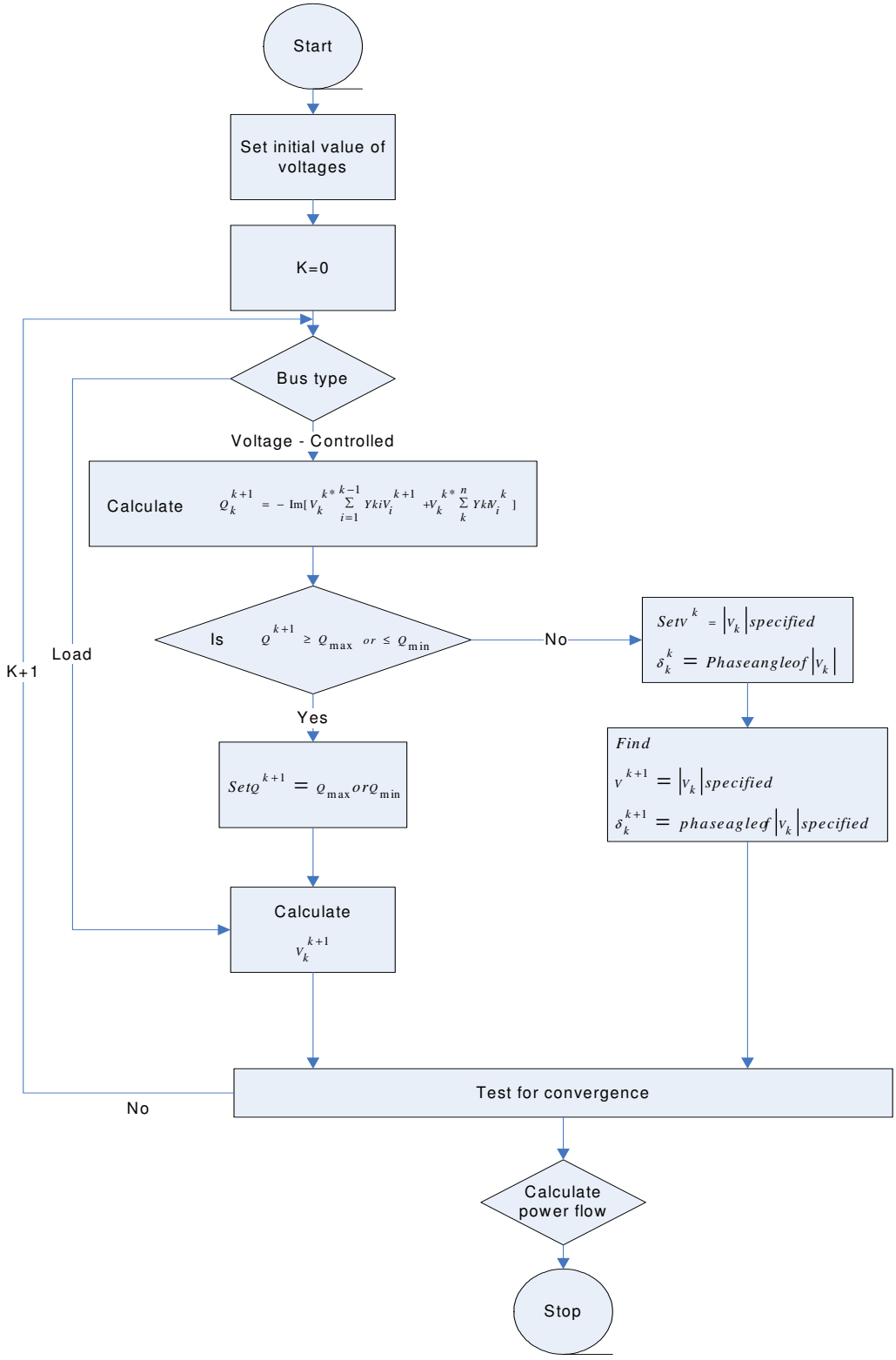


Figure 2-3 Flow Chart on Gauss Seidel method

From the same example above, the power flow and losses are as follows:

$$S12 = V1 \cdot \left[\overline{V_1 \cdot Y_{12} + \frac{(V_1 - V_2)}{Z_{12}}}\right] = 1.86 + j1.11$$

$$S21 = V2 \cdot \left[\overline{V_2 \cdot Y_{12} + \frac{(V_2 - V_1)}{Z_{12}}}\right] = -1.8 - j0.93$$

$$Loss12 = S21 + S12 = 0.0601 - j0.1803$$

$$S32 = V3 \cdot \left[\overline{V_3 \cdot Y_{23} + \frac{(V_1 - V_3)}{Z_{23}}}\right] = 0.59 + j0.54$$

$$S23 = V2 \cdot \left[\overline{V_2 \cdot Y_{23} + \frac{(V_2 - V_3)}{Z_{23}}}\right] = -0.59 - j0.52$$

$$Loss23 = S23 + S32 = 0.0057 + j0.017$$

$$S13 = V1 \cdot \left[\overline{V_1 \cdot Y_{13} + \frac{(V_1 - V_3)}{Z_{13}}}\right] = 2.1 + j1.05$$

$$S31 = V3 \cdot \left[\overline{V_3 \cdot Y_{13} + \frac{(V_3 - V_1)}{Z_{13}}}\right] = -2.05 - j0.9$$

$$Loss13 = S13 + S31 = 0.05 + j0.15$$

$$\text{Total Loss} = 0.1158 - j0.0133$$

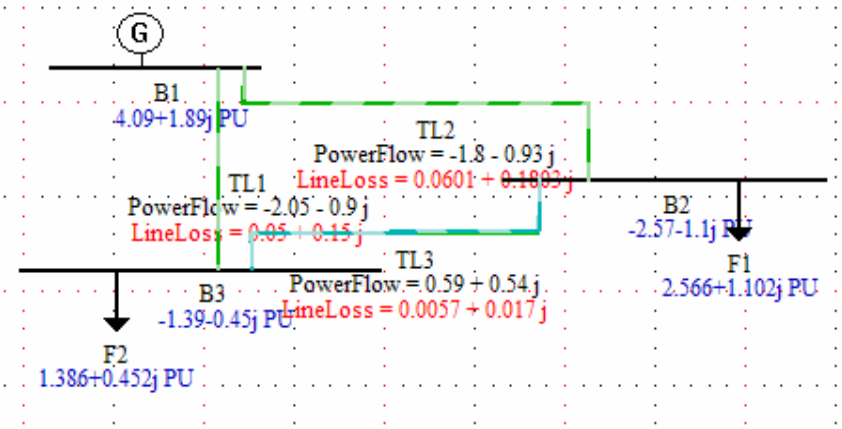


Figure 2-4 Result after gaining power flow and losses

2.7 Techniques in Solving Load Flow – Newton Raphson Method

Taylor's series expansion for a function of two or more variables is the basis for the Newton Raphson method of solving the load flow problem. In this section, discussion of the solution of a problem involving only 2 equations and 2 variables will be used, then how to extend the analysis to the solution of load-flow equations.

Given the equation of a function of 2 variables x_1 and x_2 equal to a constants K_1 and K_2 are expressed as:

$$f_1(x_1, x_2) = K_1$$
$$f_2(x_1, x_2) = K_2$$

The solution of these equations can be estimated to be $x_1^{(0)}$ and $x_2^{(0)}$. The superscripts indicate that these values are initial estimates. $\Delta x_1^{(0)}$ and $\Delta x_2^{(0)}$ are designated as the values to be added to $x_1^{(0)}$ and $x_2^{(0)}$ to give the correct solutions, so the equations become:

$$K_1 = f_1(x_1, x_2) = f_1\left(x_1^{(0)} + \Delta x_1^{(0)}, x_2^{(0)} + \Delta x_2^{(0)}\right)$$
$$K_2 = f_2(x_1, x_2) = f_2\left(x_1^{(0)} + \Delta x_1^{(0)}, x_2^{(0)} + \Delta x_2^{(0)}\right)$$

In order to solve $\Delta x_1^{(0)}$ and $\Delta x_2^{(0)}$, the above equation must be expanded in Taylor's series as given:

$$K_1 = f_1(x_1, x_2) + \Delta x_1^{(0)} \left.\frac{\partial f_1}{\partial x_1}\right|_{(0)} + \Delta x_2^{(0)} \left.\frac{\partial f_1}{\partial x_2}\right|_{(0)} + \ldots$$

$$K_2 = f_2(x_1, x_2) + \Delta x_1^{(0)} \left.\frac{\partial f_2}{\partial x_1}\right|_{(0)} + \Delta x_2^{(0)} \left.\frac{\partial f_2}{\partial x_2}\right|_{(0)} + \ldots$$

The term $\left.\frac{\partial f_1}{\partial x_1}\right|_{(0)}$ indicates that the partial derivative is evaluated for the value of $x_1^{(0)}$ and $x_2^{(0)}$.

If the partial derivatives of order greater than 1 are neglected, the above equation can be written in matrix form:

$$\begin{bmatrix} K_1 - f_1\left(x_1^{(0)}, x_2^{(0)}\right) \\ K_2 - f_2\left(x_1^{(0)}, x_2^{(0)}\right) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \Delta x_1^{(0)} \\ \Delta x_2^{(0)} \end{bmatrix}$$

$$\begin{bmatrix} \Delta K_1^{(0)} \\ \Delta K_2^{(0)} \end{bmatrix} = J^{(0)} \begin{bmatrix} \Delta x_1^{(0)} \\ \Delta x_2^{(0)} \end{bmatrix}$$

where $J^{(0)}$ is called jacobian

By finding the inverse of the jacobian, $\Delta x_1^{(0)}$ and $\Delta x_2^{(0)}$ can be determine. However, if the series expansion is truncated, the values added to the initial guess do not determine the correct solution and new estimation of $x_1^{(1)}$ and $x_2^{(1)}$ must be assumed, where:

$$x_1^{(1)} = x_1^{(0)} + \Delta x_1^{(0)}$$
$$x_2^{(1)} = x_2^{(0)} + \Delta x_2^{(0)}$$

Repeat the process until the corrections become so small that they satisfy a chosen precision index.

To apply the Newton-Raphson method to the solution of load-flow equation, bus voltages and line admittances are chosen in polar form or rectangular form. If chosen polar form, and make it into real and imaginary components with:

$$V_k = |V_k| \angle \delta_k \qquad V_n = |V_n| \angle \delta_n \qquad \text{and} \qquad Y_{kn} = |Y_{kn}| \angle \theta_{kn}$$

We have

$$P_k - jQ_k = \sum_{n=1}^{N} |V_k V_n Y_{kn}| \angle(\theta_{kn} + \delta_n - \delta_k)$$

The Newton-Raphson method is summarized in the following steps

1. Determine values of $P_{k,calc}$ and $Q_{k,calc}$ flowing into the system at every bus for the specified or estimated values of voltage magnitudes and angles for the first iteration or the most recently determined voltages for subsequent iterations
2. Calculate ΔP at every bus
3. Calculate values for the jacobian using estimated or specified values of voltage magnitude and angle in the equations for partial derivatives determined by differentiation.
4. Invert the jacobian and calculate voltage corrections $\Delta\delta_k$ and $\Delta|V_k|$ at every bus
5. Calculate new values of δ_k and $|V_k|$ by adding $\Delta\delta_k$ and $\Delta|V_k|$ to the previous values
6. Return to step 1 and repeat the process using the most recently determined values of voltage magnitudes and angles until either all values of ΔP and ΔQ or all values of $\Delta\delta$ and $\Delta|V|$ are less than a chosen precision index.

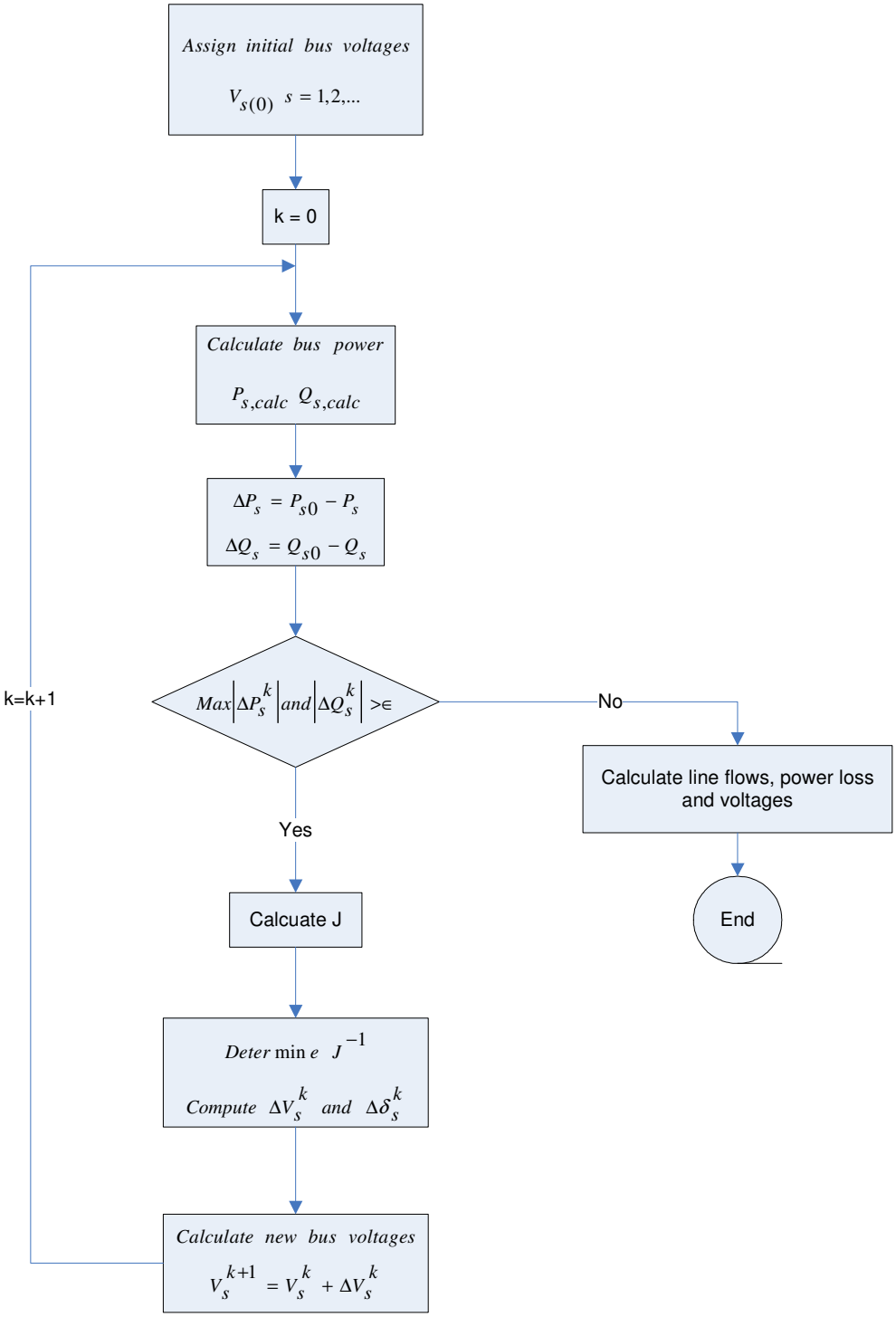


Figure 2-5 Flow chart for load bus in Newton Raphson method

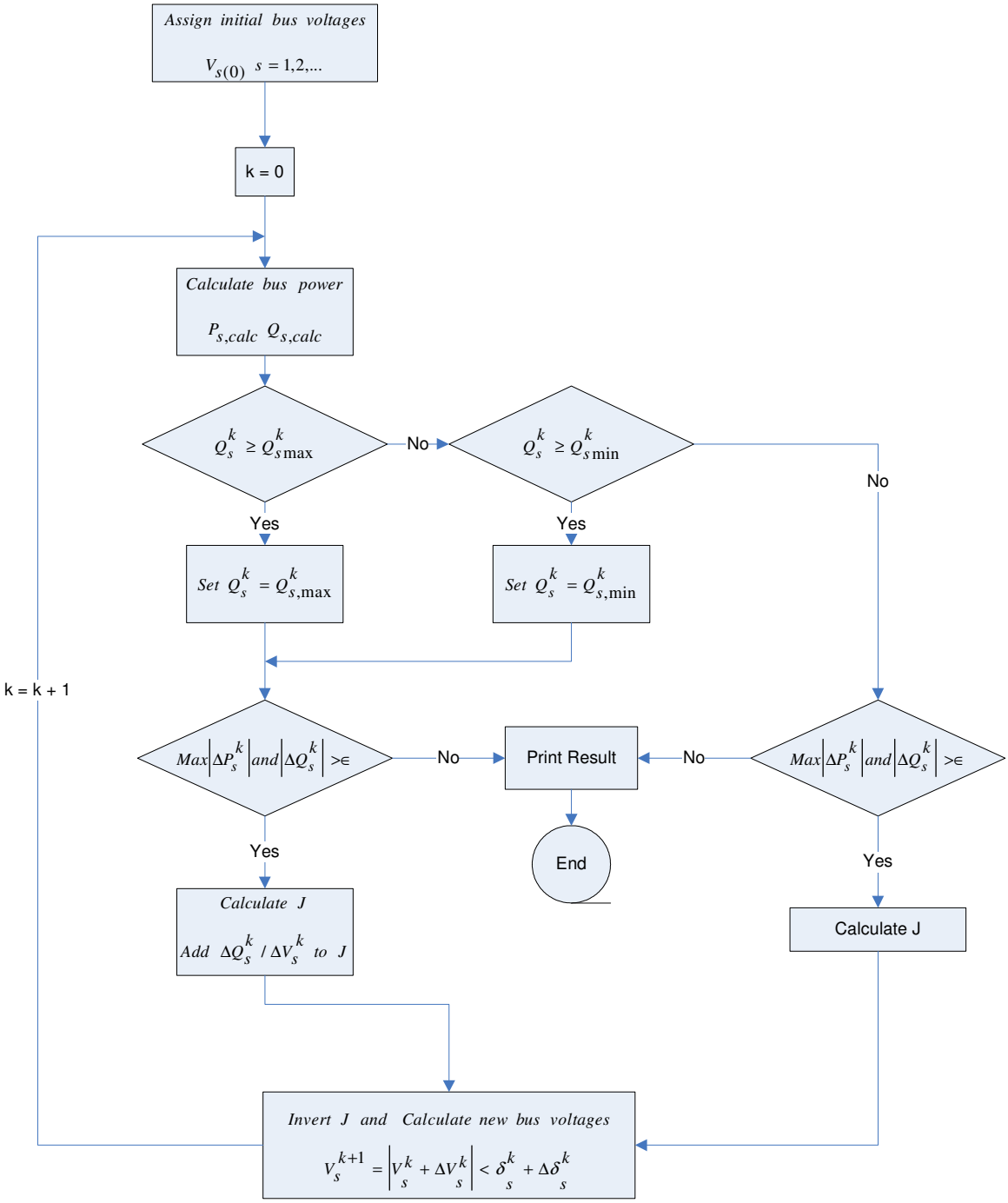


Figure 2-6 Flow chart of voltage controller in Newton Raphson method

2.8 Conclusion

Load flow analysis examines steady-state equations based on the positive definite network admittance matrix that represents the power system distribution network.

The complexity of obtaining a formal solution for load flow in a power system arises because of the differences in the type of data specified for the different kinds of buses. Although the formulation of sufficient equations is not difficult, the closed form of solution is not practical.

In this project, the Gauss Seidel Method will be used to calculate the load flow of both DC and AC section of the power system network.

Chapter 3

Visual Basic

This chapter gives detailed information on the Visual Basic Programming. It will cover the basic concepts and why Visual Basic is used in this project.

3.1 Computer Operations

All computers can carry out six operations to process data into information. Before writing any programs, it will help very much to understand these six operations. These operations are that a computer can perform are:

1. Input data
2. Store data in internal memory
3. Perform arithmetic on data
4. Compare two values and select one of two alternative actions
5. Repeat a group of actions any number of times
6. Output the results of processing

3.1.1 Input Data

For a computer to be able to transform data into information, it must be able to accept input of the data that will be processed into information. Data can be from keyboard or mouse or any other devices.

3.1.2 Store Data in Memory

Once data have been input, they are stored in internal memory. Each memory location holding a piece of data is assigned a name, which is used by the instructions to perform the processing.

3.1.3 Perform Arithmetic on Data

Once the data and instruction have been input and stored, arithmetic operations can be performed on the variables representing the data to process them into information. This includes addition, subtraction, multiplication, and division.

3.1.4 Compare Two Values and Select One of Two Alternative Actions

To so anything other than the simplest processing, a computer must be able to choose between two sets of instructions to execute. It does this by comparing the contents of two memory locations and, based on the result of that comparison, executing one of two groups of instructions.

3.1.5 Repeat a Group of Actions Any Number of Times

While all the above action can be done by normal typewriter, but this repeating action is something the computer does better than any other type of machine.

3.1.6 Output the Results of Processing

Once the processing has been completed and the required information generated, to be of any use the information must be output. It can be: displayed on a monitor, printed on paper, stored on disk, as instructions to a machine, and so on.

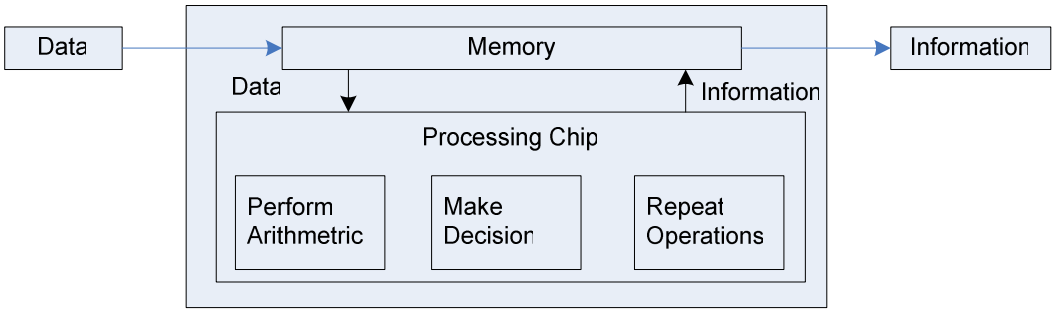


Figure 3-1 Six Computer Operations

3.2 Visual Basic

Visual Basic is a computer language that has been developed to help in creating programs that will work with the Windows operating system. It is an event-driven language that does not follow a predefined sequence of instructions. It responds to events to execute different sets of instructions depending on which event occurs, for example such as mouse click, keystrokes, or even others. In addition to being event-driven, Visual Basic has many characteristics of an Object-Oriented Language, which uses identifiable shapes, each of which has certain properties and can respond to a variety of events.

3.2.1 Visual Basic vs. Other Tools

Visual Basic can complement other tools quite well; it works well with the other tools in Microsoft's Visual Studio. Visual Basic is ideal for creating user interfaces and small – to – medium – scale applications. On the other extreme, Visual C++ can create solid ActiveX components and is a very powerful tool, but is quite difficult to use because of the overhead involved. It is typical to find Visual Basic used for the user interface and client-side development and Visual C++ used for controls, business components, and server-side applications.

Table 3- compares Visual Basic, Java-based tools (like Visual J++), and Visual BASIC. Each is a general-purpose tool to some degree.

Table 3-1 Pros & Cons of Visual Basic

Tool	Pros	Cons
Visual Basic	Easy to use Great or creating user interface Can edit code while running	Slow code
Java-Based tools	Platform independence (sort of) Strong internet capabilities	Slow code Limited functionality
Visual C++	Extremely powerful Very fast code	Difficult to use More complicated code Longer development time

3.2.2 Structure of Visual Basic File

In Visual Basic, each project has one project file with a .vbp extension and at least one form file with an .frm extension. If the project has multiple forms, then there will be an .frm file for each form in the project. In addition to the .frm files, if any forms have image or picture controls then there will also be a binary form file with an .frx file for each such form. Projects can also include module files, which are composed solely of code and have a .bas extension. Such modules are often written to be shared by multiple forms.

Finally, a type of file created by Visual Basic is the workspace file, with a .vbw extension. This file keeps track of the windows that were left open when exited Visual Basic so the next time it will start where left off.

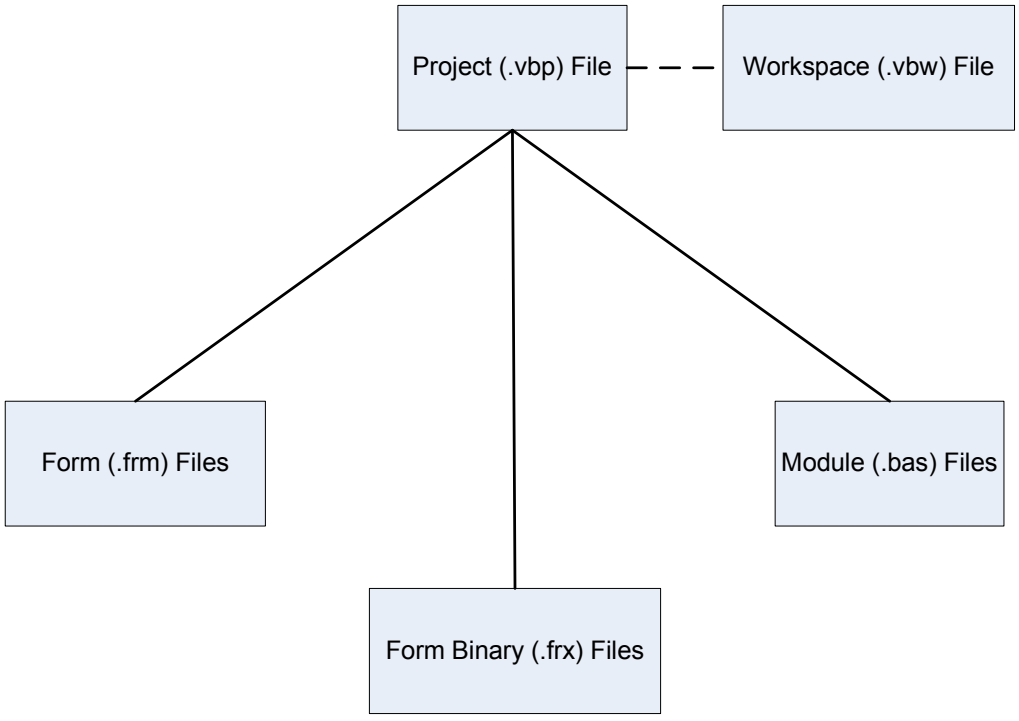


Figure 3-2 Visual Basic file structure

3.3 Version of Visual Basic

There are now three major version of Visual Basic: version 4, 5 and 6. The version of Visual Basic has been improved in many ways, but retains one of the key advantages of the original language: it is very easy to use and understand.

Version 4 was a milestone in the history of Visual Basic, since it was the first to introduce object-oriented functionality and allow the creation of 32-bit applications. However, the performance of Visual Basic 4 applications was atrocious, and very large applications were extremely difficult, if not impossible, to create.

Version 5 helped performance immensely by improving form loading and adding native code compilation. It also added a handful of new object-oriented features, changed the IDE, and began the movement to push Visual Basic into the world of web development.

Version 6, the latest version, improves server-side features and adds new wizards. It has enough characteristics of object-oriented languages that are referred as object-oriented event-driven (OOED) language.

Working with an OOED language involved combining objects with the instructions on how each object should respond to a given event. Creating an application using OOED programming language such as Visual Basic is much easier than working with a traditional programming language. Instead of having to develop the logic for the entire program, it will be easier to divide up the program logic into small, easily handled parts by working with objects and event. Series of steps to ensure correctness and completeness of the finished product:

1. Define problem
2. Create interface
3. Develop logic for action objects
4. Write and test code for action objects
5. Test overall project
6. Document project in writing

3.3.1 Define Problem

Before any development of computer application, it is absolutely necessary to clearly define the objective, that is, the problem to be solved. Only then can we begin to develop the correct logic to solve the problem and incorporate that logic into a computer application.

The problem identification step should include identification of the data to be input to the program and the desired results to be output from the program.

3.3.2 Create Interface

Once the problem has been defined, it is ready to create the interface. Creating the interface with Visual Basic is quite easy: programmers may select objects from those available and place them on the form.

3.3.3 Develop Logic for Action Objects

Once the problem has been clearly identified and the interface created, the next step is to develop the logic for the action objects in the interface. This is the step in the development process where you have to think about what each action object must do in response to an event.

3.3.4 Write and Test Code for Action Objects

Once the Visual Basic interface has been created and developed the logic for the action objects, procedures have to written in Visual Basic for each action object. This code should provide instructions to the computer to carry out one of more of the six operations listed in section 3.1, that is, input data, store data, perform arithmetic on data, compare two values and select one of two alternatives actions, repeat a group of actions any number of times, and output the results of processing.

Next will be the testing the object and correct any errors. With the interactive capabilities of Visual Basic, it is much easier to debug the code line by line. However, even if all the syntax and vocabulary are correct, the code for an object still may be incorrect – either in the manner in which it carries out the logic or in the logic itself. The best way is to use test data for which the results are known in advance. If the results for the object do not agree with the results from the hand calculations, an error exists wither in the logic or in the hand calculations. After the hand calculations have been verified, the logic must be checked.

3.3.5 Test Overall Project

Once the code for each action is tested individually, the next step is to test the overall project and correct any errors that may still exist or that may be the result of incorrect communication between objects. At this stage, it is necessary to determine whether the result obtained from the project meet the objectives outlined in the problem definition step. If the project does not meet the final user's needs, then the developer must analyze the results and the objectives to find out where they diverge.

3.3.6 Document Project in Writing

An important part of writing any computer software which a lot of programmers ignore is the documentation of the software. Documentations can be defined as the written descriptions of the software that aid users and other programmers. Documentations help users by providing instructions and suggestions on using the software. It also helps other programmers who may need to make changes or correct the programs.

3.4 Toolbox in Visual Basic

The Toolbox holds the objects that are placed on the form to create the Visual Basic project. These objects are commonly referred to as controls. The default Toolbox and the name and its control and action are show in the figure and table below.

In addition to the default Toolbar, there are standard Toolbar, which are shown in Figure 3-4 and its name, control, and action are in Table 3-2.

Figure 3-3 Default Toolbox

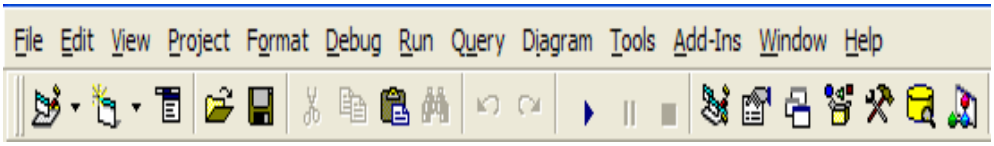
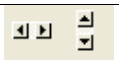


Figure 3-4 Standard Toolbox

Table 3-2 Default Toolbox Controls

Icon	Name	Action
	Pointer	Selects another control; moves controls around screen
	Picture box	Displays an image; responds to events
	Label	Displays text (read only; no input)
	Text box	Displays and input text
	Frame	Acts as container for other controls
	Command button	Responds to events
	Check box	Responds to being checked
	Option button	Responds to being on or off
	Combo box	Acts as drop-down list box
	List box	Display list of text items
	Vertical and horizontal scroll bars	Responds to scrolling by determining a value
	Timer	Determines time between events
	Drive list box	Displays a list of disk drives
	Directory list box	Displays a list of directories
	File list box	Displays a list of files
	Shape	Used to draw shapes
	Line	Used to draw lines
	Image	Displays an image on screen
	Data	Provides a link to a database
	OLE	Used for OLE (Object Linking and Embedding) operations

Icon	Name	Menu Option	Action
	Add standard EXE	File | Add Project	Add a standard.EXE project as a part of this project group (.vbg file)
	New form	Project | Add Form	Add a new form to the current project
	Menu editor	Tools | Menu Editor	Start the Menu Editor (create menus)
	Open project	File | Open Project	Open an existing project
	Save project	File | Save Project	Save current project file and all related files under same names
	Cut	Edit | Cut	Used in code editor to cut a section of code
	Copy	Edit | Copy	Used in code editor to copy a section code
	Paste	Edit | Paste	Used in code editor to paste a section of code
	Find	Edit | Find	Used in code editor to find a character or string of characters
	Undo	Edit | Undo	Used in code editor to "undo" most previous action
	Redo	Edit | Redo	Used in code editor to "redo: a previously undone action
	Run	Run | Start	Execute the current project
	Break	Run | Break	Pause execution of project to look for errors
	Stop	Run | Stop	Stop execution of project
	Project explorer	View | Project Explorer	Display project explorer window
	Properties window	View | Properties Window	Display properties window
	Form layout window	View | Form Layout Window	Display form layout window
	Object browser	View | Object Browser	Display information about available objects
	Toolbox	View | Toolbox	Display toolbox
	Data view window	View | Data View Window	Display a window in which certain types of database can be viewed and manipulated directly
	Visual component manager	View | Visual Component Manager	This tool enables you to publish, find and reuse components

Table 3-3 Toolbar Icons and Corresponding Menu Selections

3.5 Type of Project Failures

Successful projects are those that meet the functional requirements, are delivered on time, and cost no more than originally anticipated.

Most of the items in the list apply to software development in general.

1. Lack of appropriate quality assurance mechanisms
2. Poor understanding of the requirements
3. Unrealistic goals
4. Lack of or poor design
5. Arbitrary or artificial assessments of progress

3.5.1 Lack of Appropriate Quality Assurance Mechanisms

Most effective, successful quality assurance mechanisms must pervade every aspect of development. Though it is common occurrence in development, formal quality assurance cannot be relegated to the last 5 or 10 percent of the project's time, no matter how small the project. With proactive quality assurance throughout the process, programmers detect problems early, making them easier (and therefore cheaper) to fix.

Several aspects of quality assurance that are often neglected by programmers, includes:
Debugging and units testing: Although programmers may know how to break code, out of a sense of personal pride they want to avoid finding bugs. With Visual Basic, it is especially difficult to develop good debugging habits because programmers can code quickly that they do not think their work is complicated enough to merit thorough debugging.

Code reviews: code reviews are designed to provide a mechanism for ensuring that source code is up the user's standards. They also provide a great opportunity for programmers to examine one another's code and to spot potential problems before they become issues.

3.5.2 Poor Understanding of Requirements

This problem applies not only to the programmers but also to the user whom spearheads the creation of the requirements. Programmers need to have a firm grasp of what it is they are trying to do. They can miss the boat for many reasons, including poor communication. In many cases, the technical people are the ones who are ultimately responsible for the success or failure of the project. Because of this, it is crucial that programmer verify that the users really what they are asking for, so that everyone is clear. Moreover, the users often are not sure what they want, even if they claim that they do. The users may even ask for features that do not make sense technically. When this happens, it is the responsibility of the programmer to review the requirements and let the users know the potential consequences of their request.

3.5.3 Unrealistic Goal

Programmers do not make software because it is fun. It is a providers of an important service. There is an economic reason for virtually every decision made during the lifetime of a software project.

Most programmers are overly ambitious and expect to live up to unrealistic goals in the areas of time, cost, or scope. Project parameters are rarely based on estimates of how long it will really take, how much it will actually cost, or what features can really be included. Rather, these thresholds are dictated by user priorities; that is, they substantiate the needs of the user. First defines the target parameters, and then the project's estimates (time, cost, scope) are put into place to gall into line with the target. Initial estimates in the areas of time, cost, and scope are typically way out of line, but are as generous as possible from the standpoint of user needs.

Furthermore, when those creating the estimates know that Visual Basic is the tool, they may be tempted to cut the estimates. Since Visual Basic is so simple to code in, many believe development should be able to happen more quickly. Sometimes this is the case, but Visual Basic is not a silver bullet for all scheduling problems.

3.5.4 Lack Of or Poor Design

With Visual Basic, there is a strong temptation to forgo design and jump right into coding. Programmers do this because Visual Basic programming can be fun. Programmers often do this because they do not thing that Visual Basic is serious enough language for Visual Basic projects to warrant design. Whatever the reason, a lack of design can have very damaging results.

3.5.5 Arbitrary or Artificial Assessments of Progress

Most users have a more distant view of the actual status of a project than the programmers working in the trenches. Often they refer to the most highly visible aspects of the system as indication of the status of the whole project. The problem is that simple checks of the visible aspects of the system, such as the user interface or polished documentation, are misleading and are not accurate status indicators for the whole project.

To gain a realistic depiction of the status of the system, a thorough understanding of the requirements and an accurate schedule are required. Prototypes and proof-of-concepts models can help reveal weaknesses in the system and may be used as baselines for measuring the progress of the project itself.

Chapter 4

Software Development (DC Analysis)

This chapter describes the implementation aspects of the software using DC analysis developed.

4.1. Creation of Toolbox

A toolbox with various toolbars is created to provide user-friendly interface for the users. The icons within a toolbox are created using Visual Basic.

The circuit breaker icon in the Components Toolbar is used as an example to illustrate the procedure. It is drawn with a dimension of 32 X 32 pixels and is saved into two separate files (.ico file and resource file) using Visual Basic for different applications. The .ico file is used as a display in the Toolbox and the resource image is used to display on the drawing area. Figure 4.1 shows the circuit breaker drawn.

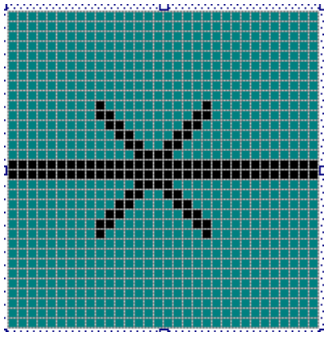


Figure 4-1 Image of Circuit Breaker

4.1.1 Components Toolbar

The components toolbar in Figure 4.2, composing of the basic components such as generators, transmission line, feeder, circuit breaker, transformers and busbar, are represented by icons, which users can click-and-drop onto the drawing area. It is created using Visual Basic.

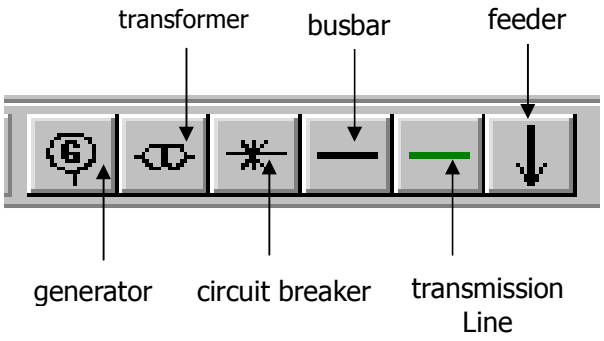


Figure 4-2 Component Toolbar

4.1.2 Simulation Toolbar

The simulation toolbar shown below provides the main simulation and animation control.

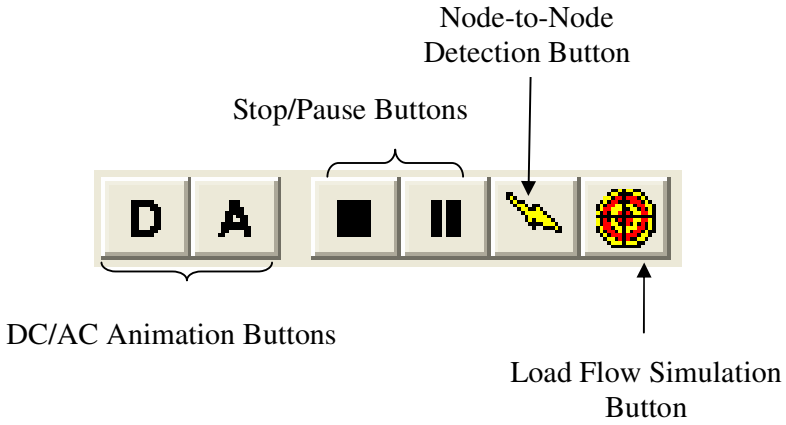


Figure 4-3 Simulation Toolbar

- **Active Model Timer**

The active model timer is responsible for the animation control of the load flow direction shown on the power system network.

- **N_to_N_Detection and Simulation_LoadFlow**

The Node-to-Node detection button performs graphical node-to-node connection checking. It calls the "N_To_N_Detection" routine of the active model.

The Load Flow Calculation button calculates load flow. It calls the "Simulation_LoadFlow" routine of the active model.

4.1.3 Standard Toolbar

The standard toolbar contains the standard window based application functions i.e. cut, copy and paste. These functions are called from a class named as "MyClipboard". This class is used to cut, copy and paste components.

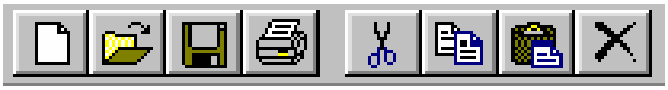


Figure 4-4 Standard Toolbar

The following are the subroutines of the sub-module, "MyClipboard".

- SaveDataFrom

These subroutines get the Database, the control ID and all the information about the control and put them into the array called MyData.

- SaveDataToDB

These subroutines get the Database and copy all the information about the control into it.

- Rotation

This function returns the rotation angle of the components.

- ToolID

This function returns the ID of the tool of which data is stored.

4.2. Using the Toolbox

4.2.1 Creation of Controls on the Form (Model)

After creating the toolbox, a drag and drop procedure is written to drag out the components onto the drawing area. The following section describes how this is done through 3 mouse-related events: Mouse Down, Mouse Move and Mouse Up in the creation and manipulation of components.

- A global variable "ButtonKey" is defined in the modCommonVariables Module. The default value is set to "Null". This variable changes when the user clicks on the Component Toolbox (e.g. for the transmission line button, the value of its ButtonKey is set to "transmissionline"). All models track this variable in the mouse move and mouse down events. All the models are informed by sharing ButtonKey variable.

a) Mouse Down Event

When ButtonKey is not Null and Mouse is clicked on the Model, a mouse down event has occurred and a component is created. The event procedure definition is as follows:

- Private Sub PicDrawing_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
- Check the ButtonKey value
- If ButtonKey = "generator" then EditGenerator flag is set to True. It means that the generator is in theUpdating phase. Then an index for the generator is found. This is done using an object called, ldGenerator. It is from a class calls LoadUnload. All generators are logically associated with it. The object ldGenerator captures the information of all generator indexes, which are created or deleted. New index can be found by invoking "NewItemNumber" method from ldGenerator. Set the value of "Current" flag to ldGenerator.NewItemNumber (Remember the current flag contains the index of the component being updated, deleted, cut, etc.) A new Label LALGenerator is also created. It is an array of labels like the Generator and is logically associated with the Generator. The caption of the LALGenerator is changed to "G" & Current + 1.

The picture of the generator is set to default. A form named as "Resource" (The statement imgGenerator(Current).Picture = Resource.Generator(1).Picture) contains all the control pictures at all angles. For angle 0 we have picture number 1; angle 90, picture number 2; angle 180, picture number 3 and; angle 270, we have picture number 4.
Feeder, Circuit Breaker and Transformer are created using the same method as mentioned above.

For busbar, it makes use of the **mouse move** event. The two end points (coordinate X1, Y1 and X2, Y2) of the busbar line are moved to the mouse coordinates in the mouse down event. The transmission line uses the same method as the busbar, with the difference that there are three logically interconnected lines controls. This subroutine terminates when the control items are created.

- If the ButtonKey is Null, the editing mode is entered. The DisableAllSquares subroutine is called, hiding all the small squares in the model while selecting a component. The mouse pointer is changed to a default value and is depicted by a Cross. All flags will be reset (Boolean Variables related to creation, deletion, updating, cut, copy and paste) to the default value i.e. No Component is being created, deleted etc. Change the flag "SaveMe" to True when a change occurs in the model.

b) Mouse Move Event

For mouse move event, the mouse icon is changed to cross if ButtonKey is not Null. If it is Null then the icon will be changed to a default mouse pointer. The function of the mouse move event is to move the selected/created component around the drawing area. The following section gives a detailed explanation of how this event works.

- The editing flag is used to determine whether the components are selected/created. If it is True, it means that one of the components is created/selected to move around in the drawing area.

- The flags associated with all the components are checked. If the flag EditGenerator is True, then the current Generator (icon), imgGenerator(Current) will move such that the mouse pointer is at its center. Here "Current" indicates the index of the current control. This is the same for other controls.

c) Mouse Up Event

This event opens the component input screen. The following provides the details.

- First check the ButtonKey. If it is not null, it means that a new component is created. The ButtonKey will then be set to null after the creation.
- The editing flag is set to false once the component is being edited. Subroutines, **OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics** are called to search for the component IDs**.** These subroutines make a search for the component with ID "CurrentControlID". If there is no such component in the database, "DB", then a new entry of the control will be created. Since the last parameter is ShowCharacteristic, it will show the input screen after a new entry. This event is terminated after completing the above tasks.
- If the ButtonKey is null, then disable all squares and reset the flags.
- When the user clicks on a component in the drawing area using right mouse button, a pop up form will be displayed. This form allows the user to enter the component parameters. These data are stored in the database for future reference.

4.2.2 Connecting Components

After creating the toolbox, the user can use it as an aid in his drawing. The user selects the desired components by clicking on the component icon in the toolbox. To place a selected icon, the user has to hold onto the left mouse button, drag the component out and place it in the drawing area.

To connect the components together, a transmission line must be used. As before, the user has to select the line icon from the toolbox. He has to first click on to the end point of a component to be connected. Then by holding on to the left mouse button, he has to drag the transmission line until it reaches the end point of another component.

After drawing the power system network, a node-to-node detection must be carried out. This function is to check whether the connections are properly done. The following section shows how to create a node-to-node detection.

4.3. Node-to-node Detection

To ensure that the power system network drawn is properly connected, a node-to-node detection routine is written. The following describes how a node-to-node detection routine was written.

All the components cover some areas on the Model except for transmission line and busbars. There are two types of components – components covering an area and components that do not cover any area.

Components that cover an area in the drawing area are Generator, Transformer, Feeder and Circuit Breaker. All these components cover an area of 32x32 pixels = 480x480 twips (1 pixel = 15 twips). But this area is not enough to check the connection. Since all the components are using different icons, connection points will vary from components to components.

Components that are not covering any area are the transmission line and busbar. Transmission lines can be connected by using their terminal points. Busbars can be connected to any component from any points.

Considering the graph theory, there may be many solutions. But the most efficient way is to traverse through all the components. A mathematical solution is required to know the areas of the components. It is first assumed that all the components are at Zero Degree.

Figure 4-5 shows how the Generator area is determined. Green spots are used to count the boxes (pixels) and Red Area is considered as a logical connection area.

The areas of the feeder, transformer and circuit breaker use the same approach.

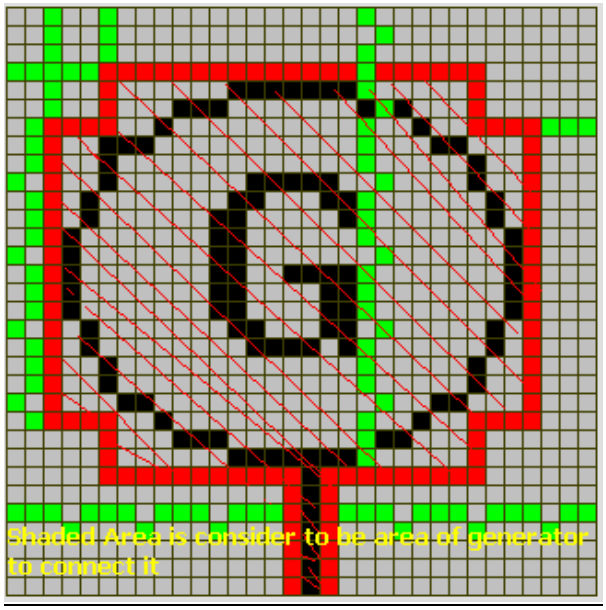


Figure 4-5 Generator

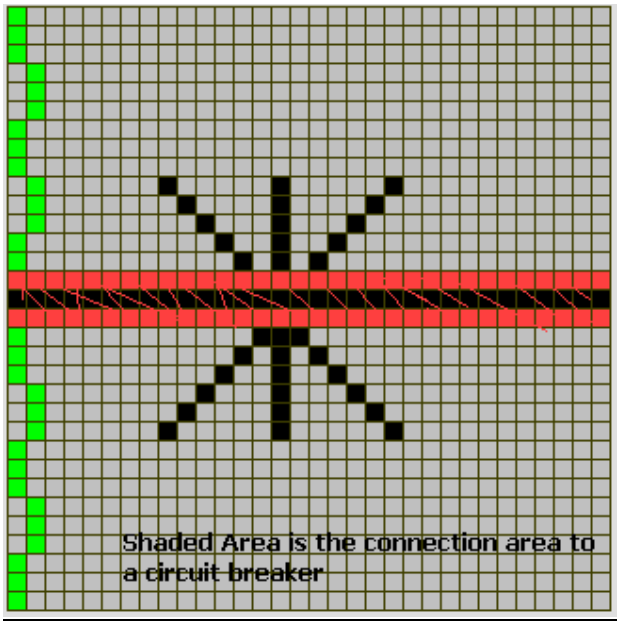


Figure 4-6 Circuit Breaker

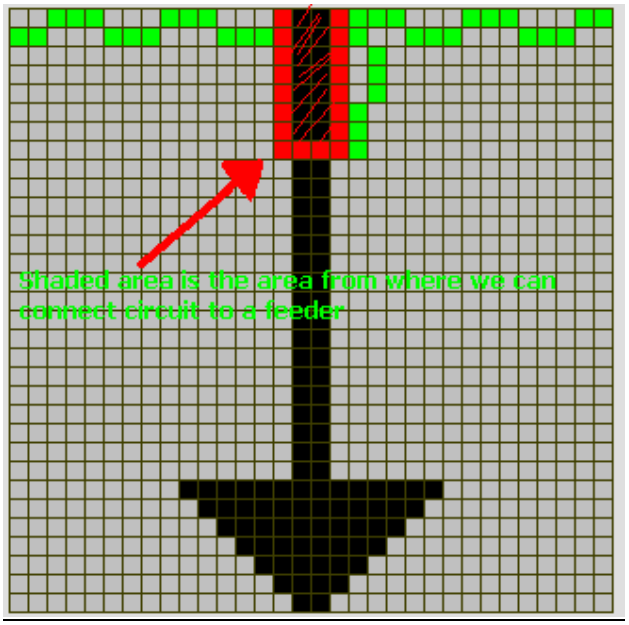


Figure 4-7 Feeder

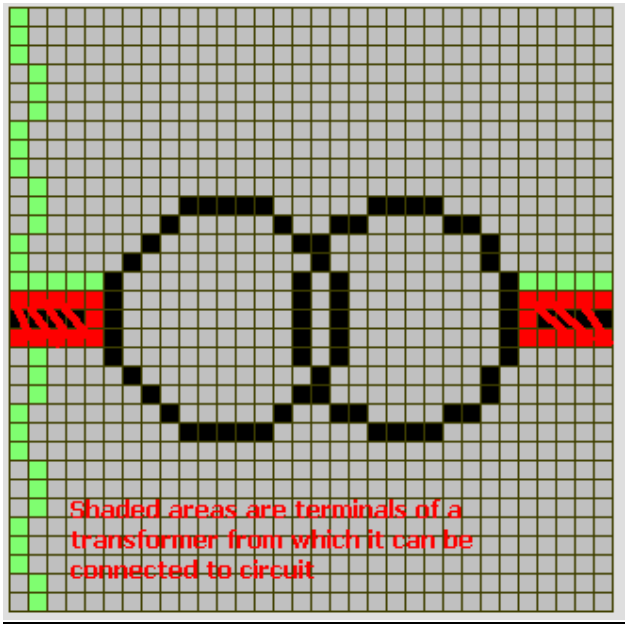


Figure 4-8 Transformer

After defining the areas of the components, a function is defined to perform the area calculation. The function is as follows:

- Private Function N_to_N_Helper(IMG As Image, X As Single, Y As Single, RotationAngle As Integer) As Boolean

This function checks if a component is connected. So for traverse in all the components, a function also interprets all the components i.e.

- Public Function N_to_N_Detection(Optional Prompt As Boolean = True) As Boolean

This function starts checking the connection of each component with all other components, e.g., Generator G1 checks its connection with other components except itself. To traverse through components, LoadUnLoad Class objects are used. Generator is used as an example to show how the connection is checked:

1. By using ldGenerator.Total, the total number of generators that can be generated on a model, e.g., if G1, G3 are on the model and G2 is deleted, the ldGenerator.Total will return a 3 instead of 2 even though G2 is deleted.

2. Using ldGenerator.IsExit(index), the component is checked whether it has been deleted. From the above example, G1, with index 1, will return true while G2, with index 2, will return a false because it is not present in the model. This is how the connections of the existing generators are checked.

Not only components that cover an area can be traversed, those that do not cover any area can also be traversed. However, for busbar there is no traversing through all components because only starting and ending points are checked. The Transition Table is searched for its existence and subsequently updated.

4.4. Load Flow Simulation Program

After simulating the node-to-node detection with no error on the connection of the model, the Load Flow simulation will be the next procedure. Load flow simulations provide the power factor of each busbar. This power factor lies in the interval -1 to 1, -1 being the worst case and +1, the ideal case. In real time systems the power factor of each busbar must approach +1 (0.98 etc.). This power factor is calculated after solving N number of linear equations dynamically.

The phase angles of the power system network are calculated using Gauss-Siedel's method. Its values are displayed on a pop up table as well as on the model itself. The values of power flow are also calculated and shown on the model. The following sections explain how load flow simulation is achieved.

4.4.1 Simulation

A simulation class is created for all simulations. This is the most important and technical class in this project. This class is highly dependent on the Transition Graph Table. The transition graph table is updated before calling the initialize procedure of this class. The simulation can be performed using the following procedure.

- A subroutine, "Initialize", is called to get the parameters that are stored previously in the database. This data is loaded into the components arrays (e.g. Busbars' data is stored into the private arrayBusbar() of type BusbarData (a structure defined inside this class)). The subroutine uses the transition graph table to locate the connection of a generator to its nearest busbar. The subroutines that play a part in building the load flow simulation are:

- Power_TransmissionLine

It returns the Induction of a specific Transmission line.

- Power_Busbar

It returns the Real Power of a specific Busbar

- Power_Feeder

It returns the real power of a specific feeder.

- Power_Transformer

It returns the real power of a specific Transformer

- Power_Generator

It returns the real power of a specific Generator

- Tran_Line_Direction

It returns the direction of the transmission line in the form of integer.
0 means no flow
1 means flow direction from (x1,y1) to (x2,y2)
-1 means flow direction from (x2,y2) to (x1,y1)

- BusBar_Net_Power

This function calculates the Net Power of a busbar (Its ID is given as parameter). It gets the information from the private arrays Generator and Feeder and calculates the Net power of the Busbar.

- Tran_Line_Power

This function gets the Busbar1_ID and Busbar2_ID as parameter and checks the transmission line, which is connected to both busbars. It returns the power of the connecting transmission line. If no transmission line is connected then it returns a 0.

- GeneratorAvalibility

It returns the Avalibility of a specific Generator

- GeneratorUnavalibility

It returns the Unavalibility of a specific Generator

- To construct the system of N linear equations and N variables, a simulation class, which creates the equations and storing the transition table in the memory is created. This class is initialized such that the database is opened with all the details of the components in it. The busbar's available supply is updated by traversing through the Transition Graph.

A function calls Public Function Tran_Line_Power (IDBusbar1 As String, IDBusbar2 As String) As Double is created. It is used to get the busbar IDs and starts traversing through the transmission lines. If it finds a connection between a transmission line and the busbars, it will return the power of that transmission line else it returns a 0.

Public Function Power_Generator(ID As String) As Double, is a function that is used for reading the information of an object Real Power. For example, it is used to get the power of a generator and other components.

Public Function BusBar_Net_Power (IDBusbar As String) As Double is used to check the connection between the feeders and the busbar. The Transition Table is also used to get the Net Power of the busbar having an ID called IDBusbar.

4.4.2 Formation of B Matrix

In mathematical terms, there are a number of ways to solve the NxN System, for example Gauss Jordan Method, Jacobi Iterative Method etc. These methods are favorable for software computation but they have some limitations. For example, Jacobi Method needs the system to be in the Diagonal Dominant form. The Inverse Matrix Method is therefore used in this computation.

The B matrix function contains a 2 dimensional dynamic array used to store data. The following subroutines are used to form the B Matrix function:

- Order

It is used to specify the order of the matrix.

- ChangeValue

This subroutine gets the row, column and values of the B matrix. It also changes the row and column values in the matrix.

- GetValue

This function returns the row and column values of the B matrix.

- Determinant

This function returns the determinant value of the matrix

- AdjointMatrix

This function returns the Adjoint Matrix of the class matrix.

- CoFactor

This function returns the cofactor of each row and column of the B matrix.

By using the above methods and subroutines, the N linear equations of the system are achieved. To form the NxN system into a matrix, a class named "Matrix" is defined. This class serves as a general matrices operation. A schematic check for all the interconnections of a busbar with other components is performed to form the NxN system. This is done using the Tran_Line_Power method and by putting the returned values into the NxN system.

4.4.3 Procedures of the Creation of Load Flow Simulation

The following procedures show how the load flow simulation is created:

- A subroutine called Public Sub Simulate_LoadFlow(Optional ShowResult As Boolean = True) is created to handle the load flow simulation. The parameter ShowResult is used at the end of this subroutine.
- Node-to-node detection is simulated first to check that the power system network is properly connected before the load flow simulation can be carried out.
- If Node-to-node detection fails to detect all component interactions, this routine will terminate by prompting a message "Please verify the node-to-node detection first! Then try again!"
- The number of busbars in the model is checked. If the number of busbars is less than 2, it terminates by prompting a message "Load Flow simulations can be performed on more than one busbars"
- A Display Form is shown on the screen while waiting for the simulation to complete.
- A Simulation Class object (Sm) is created.

- In the Simulation Class object (Sm.Initialize DB), each simulation object is initialized such that all the required data from the database is copied into the simulation object.
- All the transmission lines are checked for failure condition. The transmission line information is captured in the ldLineTransmission object.

- Check all possible transmission lines' existence. If a transmission line exists, **ldLineTransmission.IsExist(I + 1)**, then check its color. If the transmission line is red in color, it denotes that there is a fault on this line. Therefore, change the power of the transmission line in the simulation object to zero. This is done using a function calls, Sm.ZeroPower_TransmissionLine LALLineTransmission(I).
- The data used to form the Matrices is set.
- Change RowNo to zero.
- All the busbars in the drawing area are checked one by one (Busbar I). If a busbar exists, the row number (RowNo) will increase by one and the column number (ColsNo) is set to zero.
- If there is an association with all the busbars (Busbar J), the column number (ColsNo) will be increased by one. When Busbar I is not equal to Busbar J then set the row (RowNo) and the column (ColsNo) values of MatrixA to -1* using the function Sm.Tran_Line_Power (BusbarI , BusbarJ).
- If both busbars are equal, then check the association of busbar I with all others bubars (busbar K):
  - MatrixA.ChangeValue(RowNo, ColNo)= MatrixA.GetValue(RowNo, ColNo) + Sm.Tran_Line_Power(Busbar I, Busbar K)
  - MatrixB (RowNo, 1) = Sm.BusBar_Net_Power(Busbar I)
- Matrix A and B are then generated.
- The Determinant value for MatrixA is checked. If the value is zero, a message is displayed showing "Unique solution of the system is not possible" and terminates the subroutine.
- Matrix C is set to the Adjoint of A.
- (Set C = MatrixA.AdjointMatrix)
- Matrix D is set to the Product of C and MatrixB.
- The Grids on the form frmSimulationLoadFlowResult are arranged according to the Orders of the MatrixA, MatrixB and D. It means the number of rows and columns in the Grids and Matrices are the same.
- The values are entered into the rows and columns of the grids.
- Create and Display an Information label (LalInfo) with its caption to real power of a component.

4.5. Animation

After the load flow simulation, the user can view the load flow direction by clicking on the Animation Play Button in the Simulation Toolbox.

A timer control, "TimerSimulationFlow", is used to display the animation since the information about the load flow direction in the AnimationTLS array is known. The following shows the procedure of incorporating animation.

- The values in the AnimationTLS array are checked.
- Checking AnimationTLS at index I.
- If the value is 0, transmission line with index I will not be animated.
- If the value is 1 then start the animation. This animation has 3 parts. They are:
  - The starting part of the transmission line called from the function, LineTranStart.
  - The mid range of the transmission line is animated from LineTranMid.
  - The ending part of the transmission line called from LineTranEnd.

- If the value is –1, then end the animation from LineTranEnd, LineTranMid and LineTranStrat.
- The direction of the load flow can be determined by looking into the Transition Graph Table. If the transmission line is connected from Busbar I to Busbar J then set θ_1 to Power of Busbar I and set θ_2 to Power of Busbar J.

When (θ_1- θ_2) is zero, it means no flow. Then set the AnimationTLS to zero. If not, then check the equation -- (θ_2 - θ_1)/Power of Transmission line. If the result calculated from the equation is negative, the flow direction will be negative with that, another check is inserted, that is to check if the power flow is smaller than the transmission line capacity limit, if it's exceeded, AnimationTLS = 2 else AnimationTLS = 1. If the result is positive, then the flow direction will be positive, then checking the capacity limit, if exceeded, AnimationTLS = -2 else AnimationTLS = -1. At the end, the system will enable the animation

```
TimerSimulationFlow.Enabled = True
```

Figure 4-9 shows an example of the animation of load flow direction.

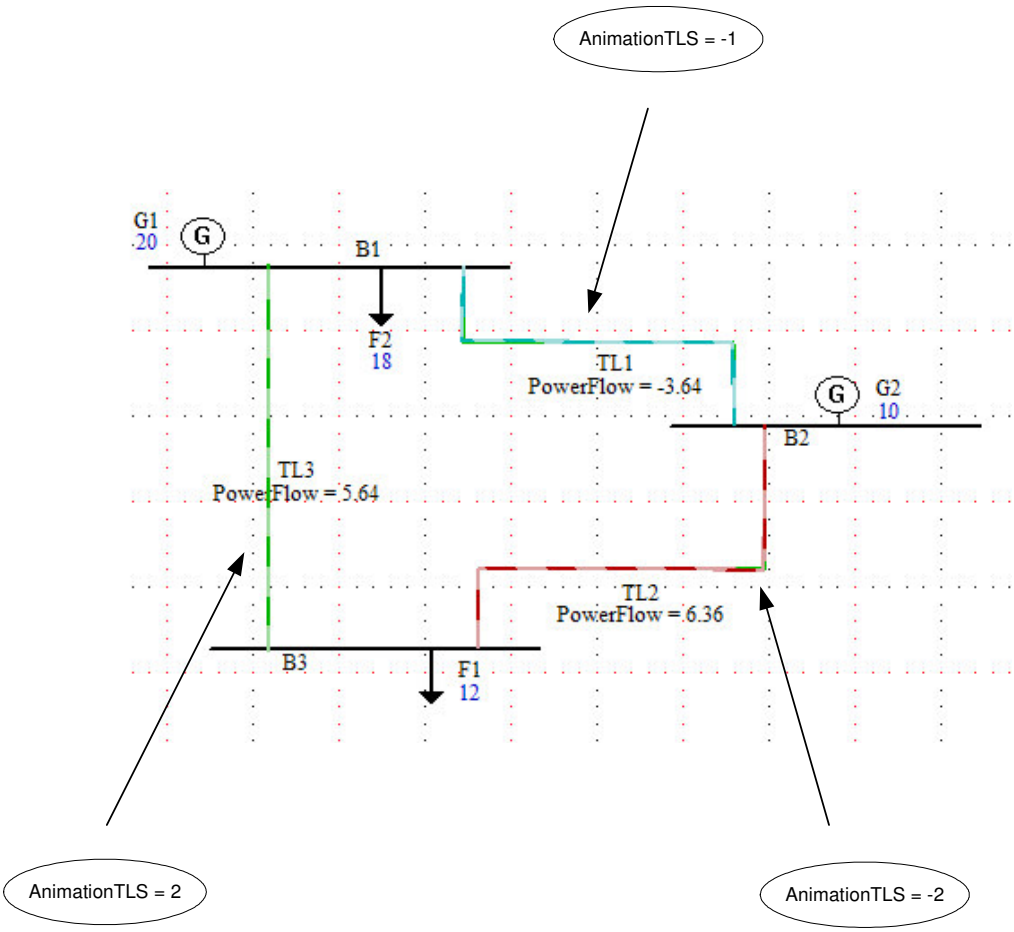


Figure 4-9 Animation of the Load Flow direction

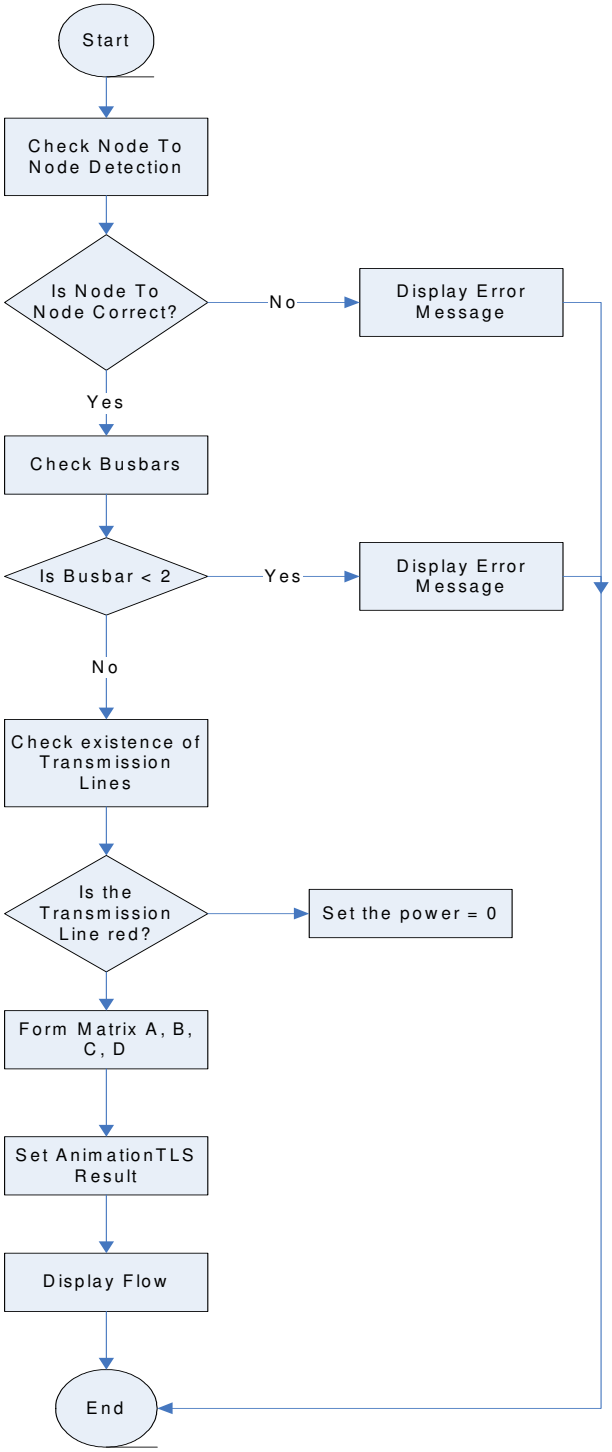


Figure 4-10 Flow chart on the Overall simulation of DC analysis

Chapter 5

Software Development (AC Analysis)

This chapter describes the implementation aspects of the software using AC analysis developed. In particular, the following are reviewed: Bus Admittance, Gauss Seidel Method, Line Flow, Line Losses and Animation of power flow.

AC Gauss Seidel Method consists of four parts. The main portion will be the Gauss Seidel method calculation, which is after Bus Admittance, followed by calculation of the line flow and losses, the last part will be the displaying of the animation flow.

5.1 Data Preparation

Before getting into the real simulation, there are two data which the simulation will need. They are the Bus Data File and Line Data File. According to the name, it is quite obvious what each file consists of. Bus Data File consists of information related to the bus bars while Line Data File consists of information related to the transmission lines.

The initial step in the preparation of input file is the numbering of each bus. With the buses being numbered, it will be easier to collect the data of each individual bus into the Bus Data File. It's assigned to the bus bar when it was first drawn. Only when it is deleted, then another drawn bus bar will get the number of the deleted bus bar.

The format for the bus entry is chosen to facilitate the required data for each bus in a single row. The information required must be included in a matrix called busdatamatrix.

5.2 BusDataMatrix

Column 1 : Bus number
Column 2 : Bus Code
Column 3 : Voltage Magnitude in PU
Column 4 : Voltage phase angle in degrees
Column 5 : Load power in MW
Column 6 : Load power in MVAr
Column 7 : Generator power in MW
Column 8 : Generator power in MVAr
Column 9 : Generator minimum power MVAr
Column 10 : Generator maximum power MVAr
Column 11 : Injected MVAr of shunt capacitors

The bus code entered in column 2 is used for identifying load, voltage-controlled and slack buses as outlined below:

1 Slack bus
0 Load bus. Initial voltage is being estimated. It is usually 1 for the voltage magnitude and 0 for the phase angle. In the later part, it will be calculated by using the Gauss – Seidel method.
2 Voltage-Controlled bus.

5.3.1 Creation of BusDataMatrix

According to the information mentioned above, the busdatamatrix required 11 columns and rows will be the number of bus bars in the diagram. Therefore busdatamatrix is set with the number of bus bars as number of rows, and with 11 columns by:

```
busdatamatrix.Order ldLineBusbar.totalloaded,11
```

Before putting the data into the busdatamatrix, a check on the existence of bus bars is required, by using:

```
ldLineBusbar.IsExitst(I+1)
```

With that, each individual column's information will be stored in.

First columns will be a running number, as long as a bus bar is found; the number will increase by one.

Second and tired columns are taken from the Simulation procedures, by using the command:

```
sm.buscode_busbar(LALLineBusbar(I)
Sm.Voltage_Busbar(LALLineBusbar(I)
```

Fourth column is the phase angle of voltage, but in the system we did not create any angle for the bus bar, therefore assume that all buses having zero phase angle. So by default, set it to zero.

Fifth to six columns required some connection with the load, therefore there is need to communicate with the database to retrieve the load (feeder) information by:

```
Set rsForAll = DB.OpenRecordset("Feeder Parameters", 2, 0)
```

Check if the feeder in the database is connected to the bus bar, if it is, get the real power and reactive power by:

```
sm.Power_Feeder(LALFeeder(f)
sm.Reactive_Feeder(LALFeeder(f)
```

Seventh to tenth columns required information from the generator, therefore same as column five to six, communication with the database is required. Communication is done by:

```
Set rsForAll = DB.OpenRecordset("Generators", 2, 0)
```

A check is done in this database, to see if this generator is connected to which bus bar, if it is connected to the specify bus bar, then get the real, reactive, minimum and maximum power by:

```
sm.Power_Generator(LALGenerator(f))
sm.Reactive_Generator(LALGenerator(f))
sm.Qmin_Generator(LALGenerator(f))
sm.Qmax_Generator(LALGenerator(f))
```

And the last column is set to zero, as we have not incorporated in the injected MVAr of the shunt capacitors.

5.3 LineDataMatrix

Column 1 : Bus number (Connected From)
Column 2 : Bus number (Connected To)
Column 3 : Transmission line resistance
Column 4 : Transmission line reactance
Column 5 : One half of the total line charging susceptance in PU
Column 6 : Transformer Tap setting for lines. It should be set to "1".

5.3.1 Creation of LineDataMatrix

We can see that linedatamatrix required 6 columns, and the number of rows depends on the number of connection of the transmission lines. Therefore, we set it like this:

```
linedatamatrix.Order count, 7
```

Where count = the number of combination of transmission lines in the database.

Since linedatamatrix takes all the data from transmission lines, therefore, we opened a connection with the database, transmission lines table:

```
Set rsForAll = DB.OpenRecordset("Transmission Lines", 2, 0)
```

Same as the busdatamatrix, linedatamatrix must check for the transmission lines existence by:

ldLineTransmission.IsExist(I + 1)

After done the above, it is time to collect data from the database.

First column contains the bus number from the database, transmission lines – "connected from".

rsForAll.Fields("Connected From")

Second column is the same as first column, but it contains another information from the database, which is the "connected to".

rsForAll.Fields("Connected To")

Third column will be the real power of the transmission line, also from the database, but this time round it is according to the transmission line ID instead of the first and second column, they follow the sequence of the database. For example,

Table 5-1 Database of Transmission Line (Part 1)

TransmissionLinesID	Connected Fron	Connected To
TL2	B1	B5
TL3	B2	B3
TL1	B1	B2
TL5	B3	B5
TL6	B5	B4
TL4	B3	B4

The first row shows "Connected From" = B1 and "Connected To" = B5 which is saying this transmission line is connected from Bus Bar 1 to Bus Bar 5. By looking at the same row, under TransmissionLinesID, it reads TL2, which is transmission line 2. As such, a variable, 'transIndex' is created to keep track of this transmission line ID. With that, third column's information can be retrieved even though the cursor is still at first row. It makes use for the sm – simulation in the procedure and of course, the variable 'transIndex'.

```
sm.Reactance_TransmissionLine(transIndex)
```

As for fourth column, it is meant for the reactive power from the transmission lines database same as column 3, therefore by making use of the same concept. The information for this column is taken out.

```
sm.Imagine_TransmissionLine(transIndex)
```

Column five and six are pre-default value. Fifth column will be under further implementation, while sixth column is set to "1" as according to the requirement.

The last column an extra column is to record down the transmission ID number. Transmission ID consists of "TL" and a numbering. For this, we only need the numbering behind the "TL", therefore we make use of the "Right" and the "Len" command in the Visual Basic.

```
Right(transIndex, Len(transIndex) - 2)
```

5.4 Bus Admittance

This portion of code requires the line and transformer parameters and transformer tap settings which will be recorded as linedatamatrix in the source code. It converts impedances to admittances and obtain the bus admittance matrix.

First of all, collect all the impedances into a matrix, zrect. Since all the information is recorded in busdatamatrix and linedatamatrix, most of the time from now on will get the data from these matrixes.

Zrect will get information from linedatamatrix, column 3 and column 4 as its real and imaginary terms of the impedances.

Zrect = linedatamatrix(column 3) + linedatamatrix(column 4) j

After getting the information for Zrect, now it is time to form the YBus matrix. With the Zrect matrix, Yrect matrix can be formed by taking an inverse of Zrect. YBus matrix collects the admittances from the Yrect matrix according to the linedatamatrix column 1 and 2 in sequence.

Table 5-2 Database of Transmission Line (Part 2)

TransmissionLinesID	Connected Fron	Connected To	TotalInductance	ResistancePerKm
TL2	B1	B5	0.126	0.031
TL3	B2	B3	0.126	0.031
TL4	B3	B4	0.336	0.084
TL5	B3	B5	0.21	0.053
TL6	B5	B4	0.252	0.063
TL1	B1	B2	0.168	0.042

For this example above, YBus(1,2) will be having the value of $(-0.031 - 0.126\ j)^{-1}$ and YBus(2,1) will have $(0.031 + 0.126\ j)^{-1}$, which is YBus(1,2) = -1.84 + 7.48 j and YBus(2,1) = 1.84 – 7.48 j.

After which, the diagonal elements of the YBus matrix will be formed. To get this correctly done, two *FOR* loops are used. External *FOR* loop, loop for the number of bus bars, while the internal *FOR* loop, loop for the number of combination in the transmission lines database, by making use of the variable, "count". On top of that, there is another *IF* statement.

The first loop which is the internal one, loop according to the variable "count", it will keep adding the Yrect (J) amount into YBus(I) as long as the linedatamatrix column 1 or 2 is equal to variable "I". Where variable "I" is the external loop and variable "J" is the internal loop.

From the above example, YBus(1,1) will have the amount Yrect(1,5) + Yrect(1,2) which will be 3.24 – 13.09 j. In short, it is adding the admittances which are connected to the bus.

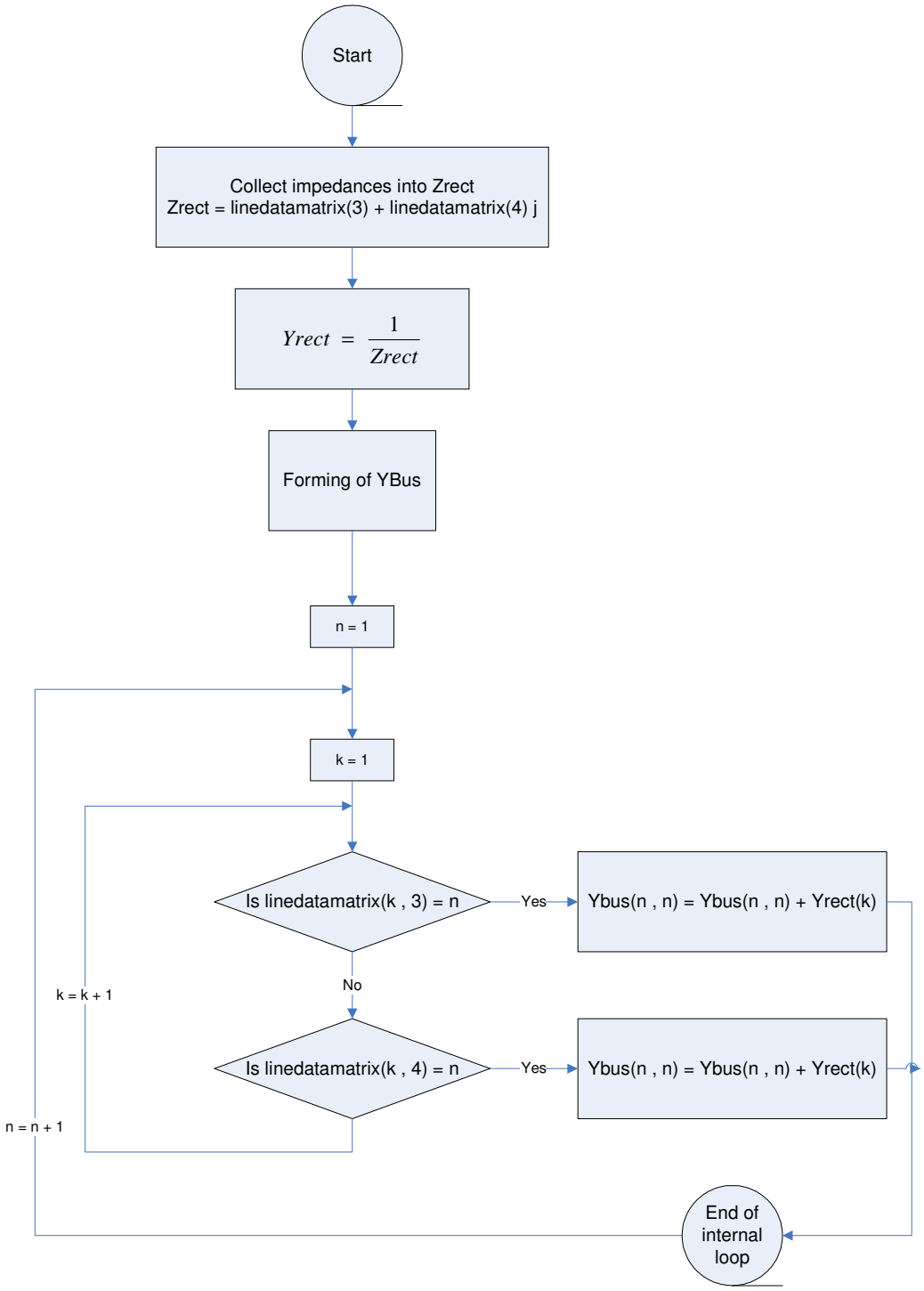


Figure 5-1 Flow chart on forming of YBus

5.5 Gauss Seidel Method

This portion obtains the power flow solution by the Gauss Seidel method and required the busdatamatrix and linedatamatrix. It is designed for the direct use of load and generation in MW and MVAr, bus voltages in PU, and angle in degrees. Loads and generation are converted to PU quantities on the base MVA selected (100MVA). A provision is made to maintain the generator reactive power of the voltage-controlled buses within their specified limits. The violation of reactive power limit may occur if the specified voltage is either too high or too low. After a few iterations, the VAR calculated at the generator buses are examined. If a limit is reached, the voltage magnitude is adjusted in steps of 0.5 percent up to ± 5 percent to bring the VAR demand within the specified limits.

In order to perform power flow analysis by the Gauss Seidel method, a few variables are needed, they are:

accuracy	:	Power Mismatch Accuracy
accel	:	Acceleration Factor
maxiter	:	Maximum Number of iterations
basemva	:	Fixed at 100MVA in program

5.5.1 Fulfils Requirement of Iterations

As we know, Gauss Seidel method will have a lot of iterations. We got to state how many iterations by letting the system meet the following requirements before getting into the loop for the whole program:

While maxerror >= accuracy And iter <= maxiter

maxerror which is default value as "10" must be greater than or equal to accuracy which is 0.001 by default, and iter (iterations) which is set to "0" must be smaller than or equal to maxiter (maximum iterations) which is "100".

With that being fulfilled, each loop will start with an increment in iteration, which means, the first loop will have iteration = 1. Starting with a *FOR* loop. The number of loop depends on the number of bus bars, which is kept in the variable "n".

5.5.2 Current and Apparent Power

With a *FOR* loop, the number of loops depends on the number of combination in the transmission line database, which is kept in the variable "L".

In each loop, the system will compare the whether the bus bar (Connected From) or (Connected To) is equal to the variable "n", if it is equal, it will add the current of the bus bar into another variable "YV". The current of the bus bar is calculated with the help of Ybus matrix and the voltage of the bus bar.

YV = YV + Ybus("Connected From", "Connected To") x Voltage of "Connected To"

YV = YV + Ybus("Connected To", "Connected From") x Voltage of "Connected From"

This will continue until the number of transmission lines combination has been used up.

Since we got the currents of the individual bus bar, the apparent power can be found too by using the following equation:

SC1 = conjugate(Voltage of bus bar "n") x (Ybus(n , n) * Voltage of bus bar "n" + YV)

SC2 = conjugate SC1

5.5.3 Bus Code 1

This is a slack bus, which all the voltage and phase angle are known, therefore under this portion, it is just assigning the voltages and phase angle into the variables created, "P", "Q", "DP", "DQ" and "VC".

5.5.4 Bus Code 2

This is a voltage-controlled bus, where we have to test the MVAr of generator after 10 interations. If it is not within limits, Vm(n) will be changed in steps of 0.005 PU up to 0.05 PU in order to bring the generator MVAr within the specified limits.

```
Q(N, 1) = screct (N, 2)
If busdatamatrix (N, 10) <> 0 Then
        qgc = q (N, 1) * 100 + qd (N, 1) - busdatamatrix (N, 11)
        If Abs(dq (N, 1)) < 0.005 And iter >= 10 Then
                If dv (N, 1) <= 0.045 Then
                        If qgc < busdatamatrix (N, 9) Then
                                vm (N, 1) = vm (N, 1) + 0.005
                                dv (N, 1) = dv (N, 1) + 0.005
                        ElseIf qgc > busdatamatrix (N, 10) Then
                                vm (N, 1) = vm (N, 1) - 0.005
                                dv (N, 1) = dv (N, 1) + 0.005
                        End If
                End If
        End If
End If
```

However, since magnitude of voltage is specified, only the imaginary part of the iteration voltages is retained, and its real part is selected in order to satisfy:

$$vcr = \sqrt{vm(n)^2 - vci^2}$$

Where vcr and vci are the real and imaginary components of the iterations voltage in sequence.

The rate of convergence in increased by applying an acceleration factor to the approximate solution obtained from iteration.

vreal (N, 1) = vreal (N, 1) + 1.3 * (vcrect (N, 1) - vreal (N, 1))

where 1.3 is the acceleration factor. Its value is default in the system.

The previous values of voltages will be replaced by the updated voltages immediately in the subsequent equations. The process is continues until changes in the real and imaginary components of bus voltages between successive iterations are within a specified accuracy.

5.5.5 Bus Code 0

This is a load bus where estimation of the initial voltage are entered in. it will be solved for the real and imaginary components of voltage same as what was done in bus code 2.

$$vcr = \sqrt{vm(n)^2 - vci^2}$$

vreal (N, 1) = vreal (N, 1) + 1.3 * (vcrect (N, 1) - vreal (N, 1))

Once the solution is converged, the new real and reactive powers at the slack bus and reactive powers at the voltage controlled bus wil be computed:

pg (N, 1) = p (N, 1) * 100 + pd (N, 1)
qg (N, 1) = q (N, 1) * 100 + qd (N, 1) - busdatamatrix (N, 11)

where 100MVA is the basemva by default in the system

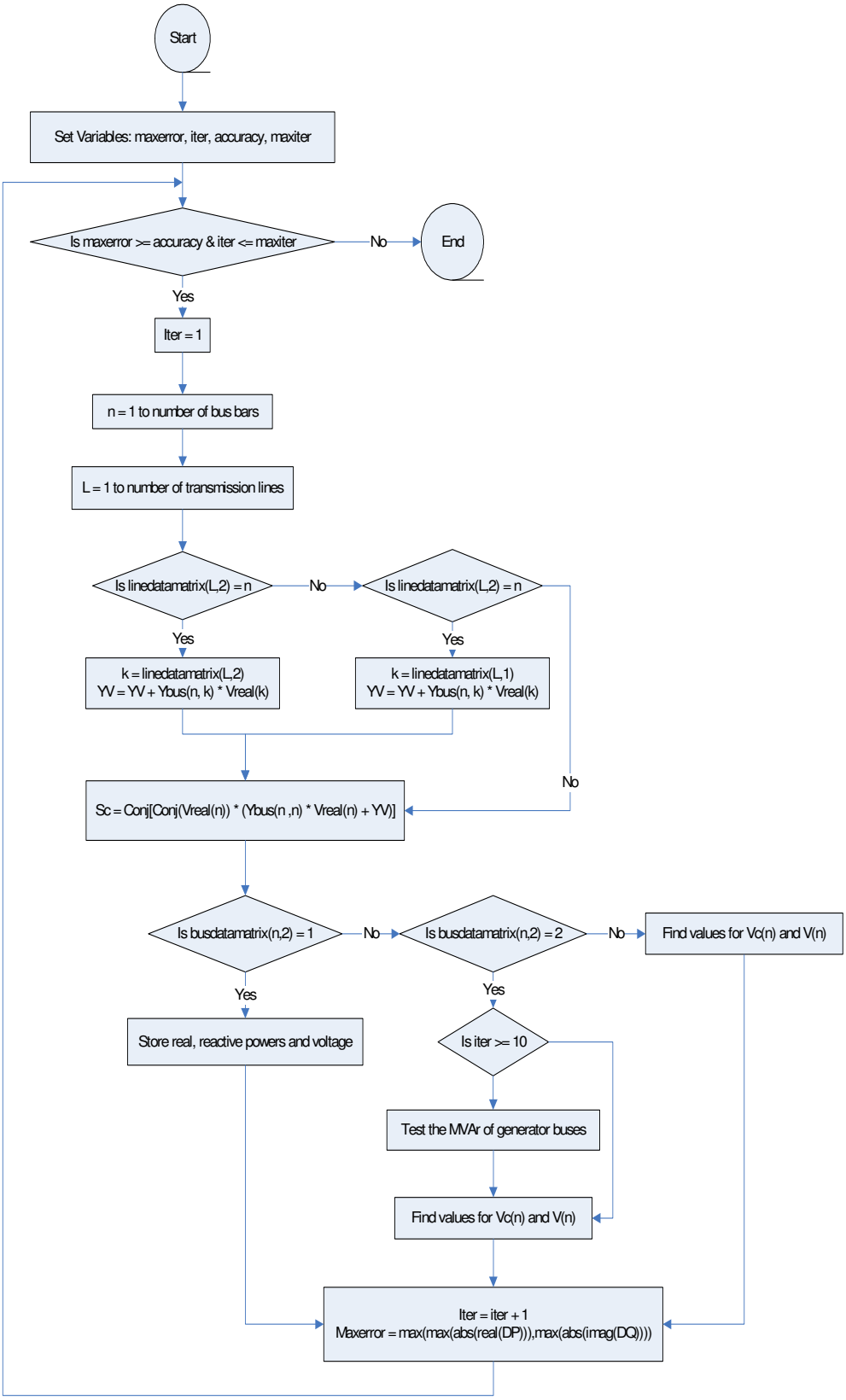


Figure 5-2 Flow chart on Gauss Seidel method

5.6 Line Flow and Losses

After the iterative solution of bus voltages, the next step is the computation of line flows and line losses.

5.6.1 Current and Apparent Power

With a **FOR** loop, the number of loops depends on the number of bus bars, which is kept in the variable "n" in this **FOR** loop, there is another **FOR** loop, the number of loops depends on the number transmission lines, which is kept in the variable "L".

In each loop, the system will compare the whether the bus bar (Connected From) or (Connected To) is equal to the variable "n", if it is equal, it will calculate the line current of transmission line connected from and to, "In" and "Ik". Next it will calculate the apparent power of the transmission line, "Snk" and "Skn"

In = (V(n) – V(k)) * Y(L)
Ik = (V(k) – V(n)) * Y(L)
Snk = V(n) * conjugate (In) * 100
Skn = V(k) * conjugate (Ik) * 100

In order to calculate the line losses, simply add the two apparent powers together.

SL = Snk + Skn

This will continue until the number of transmission lines combination has been used up.

5.7 Display and Animation

The direction of the load flow can be determined by looking at the real power of the transmission lines. If the real power of transmission line connected from Busbar I to Busbar J is positive then the system will display at the transmission line. The system will also take note of the (Connected From) and (Connected To), if positive power is from (Connected From) then set the AnimationTLS = "-1" or else set it to "1"

```
If snkrect (1, 1) > 0 Then
   If linedatamatrix (l, 1) = N Or linedatamatrix (l, 2) = N Then
      Load LALPowerFlow(l - 1)
      If snkrect (1, 2) < 0 Then
         LALPowerFlow(l - 1) = snkrect (1, 1) / 100 & " - " & Abs(snkrect (1, 2)) / 100 & " j"
      Else
         LALPowerFlow(l - 1) = snkrect (1, 1) / 100 & " + " & Abs(snkrect (1, 2)) / 100 & " j"
      End If
      Load LALlineloss(l - 1)
      LALlineloss(l - 1) = "LineLoss = " & slrect (1, 1) / 100 & " + " & slrect (1, 2) / 100 & " j"
      transnum = linedatamatrix (l, 7)
      LALPowerFlow(l - 1).Visible = True
      LALlineloss(l - 1).Visible = True
      If linedatamatrix (l, 1) < linedatamatrix (l, 2) Then
         AnimationTLS(l - 1) = 1
      Else
         AnimationTLS(l - 1) = -1
      End If
   End If
End If
```

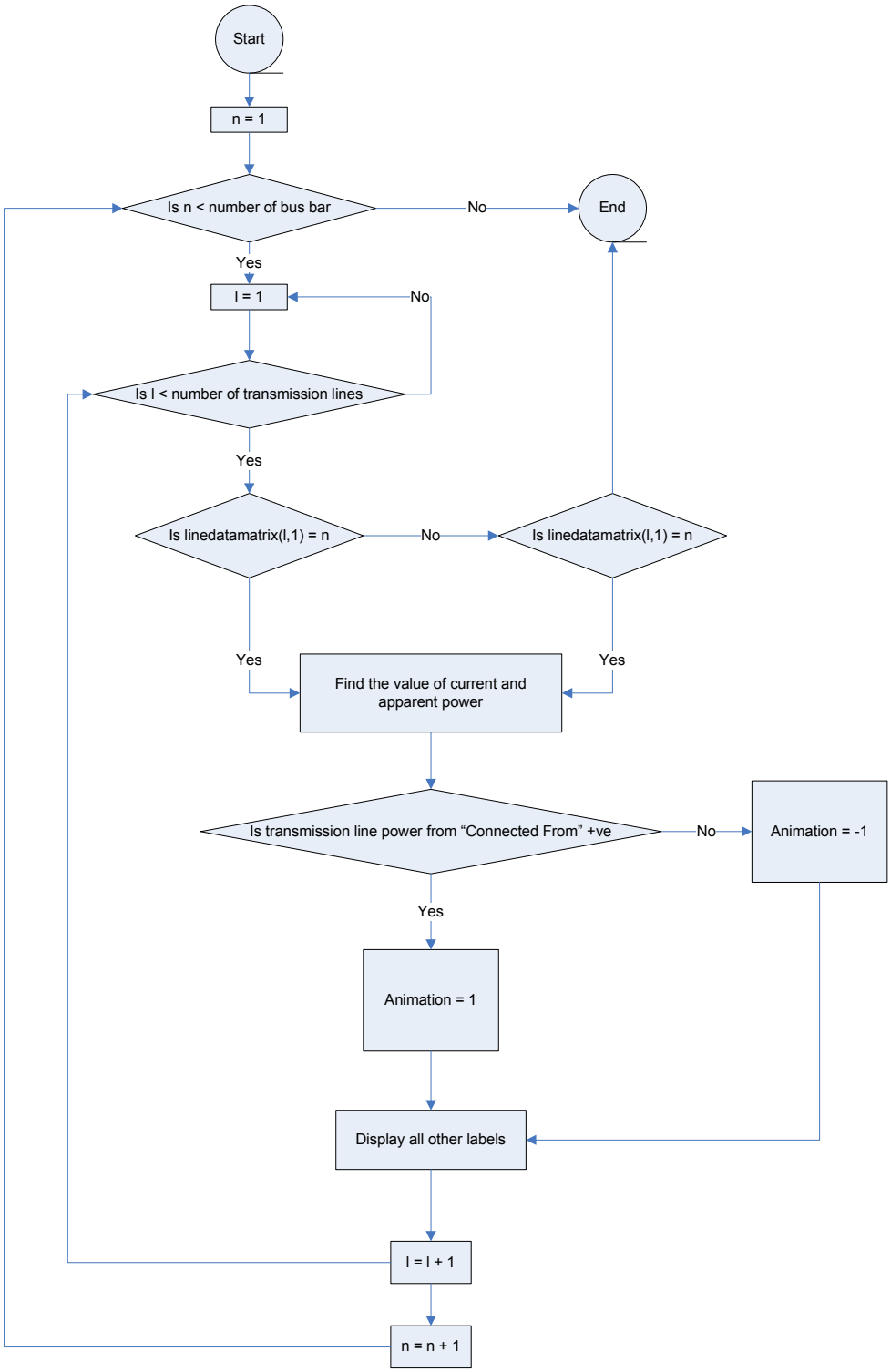


Figure 5-3 Flow chart on Animation Display

Chapter 6

Overall Simulation of RBTS system

6.1. Overview of Software

Power system Load Flow calculation is defined as the ability of the system to provide an adequate supply of electrical energy. In this software – Software Development for Power System Analysis, the Load Flow analysis was studied and analysed.

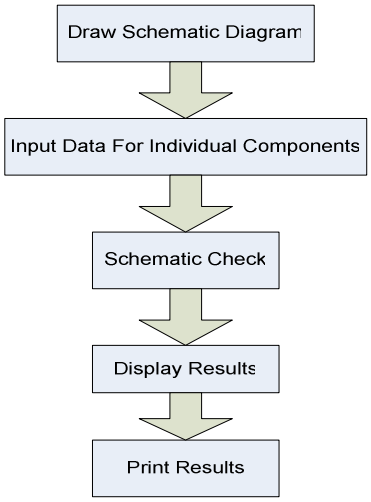


Figure 6-1 Software Flow

6.2. System Information

RBTS (Roy Billinton Test System) was developed at the University of Saskatchewan by the power systems research group. It has been used to compare and test a wide range of generating capacity and composite system evaluation techniques and subsequent digital computer programs. It has proved to be extremely valuable in highlighting and comparing the capabilities of programs used in reliability studies, the differences in the perception of various power utilities and the differences in the solution techniques.

6.3 RBTS schematic diagram

With the developed software, a powerful drawing toolbar is created. The developed software can be tested using the RBTS schematic diagram

A structured RBTS schematic diagram is drawn with the toolbar shown in Figure 6-3. User will key in their input data for each component as shown in Figure 6-2.

The user can key the real power and nominal voltage for the component. The date installed is selected using the calendar built in. The maintenance interval is keyed in by the user and the next maintenance date is computed by the software automatically.

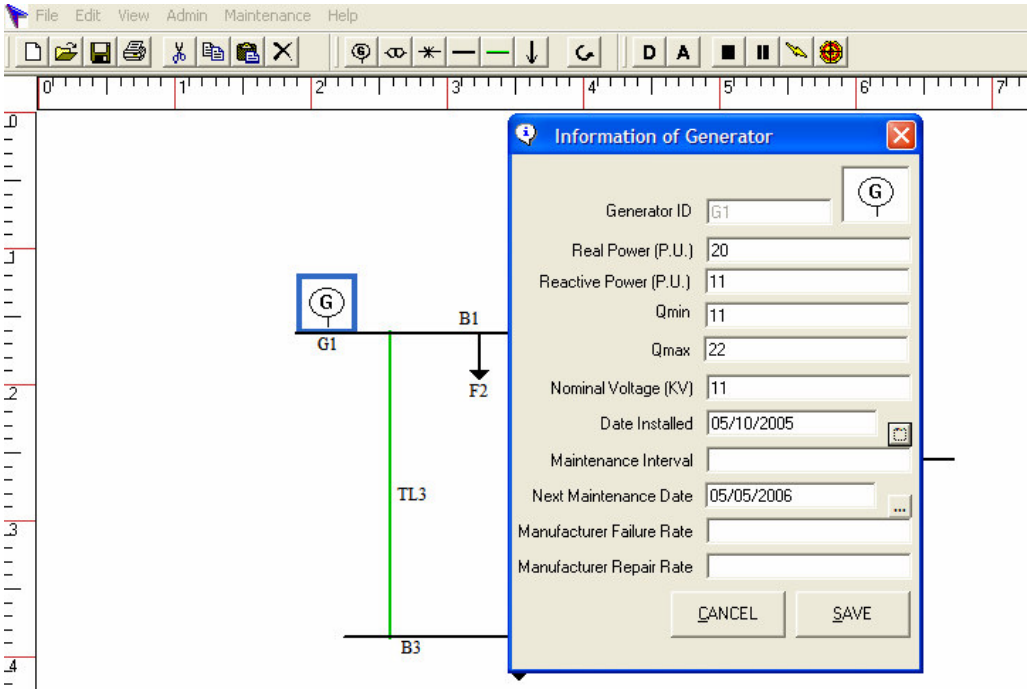


Figure 6-2 User Input for Generator

The system in Figure 6-3 has eleven generating stations each accompanied with a transformer and circuit breaker, four load buses, nine transmission lines.

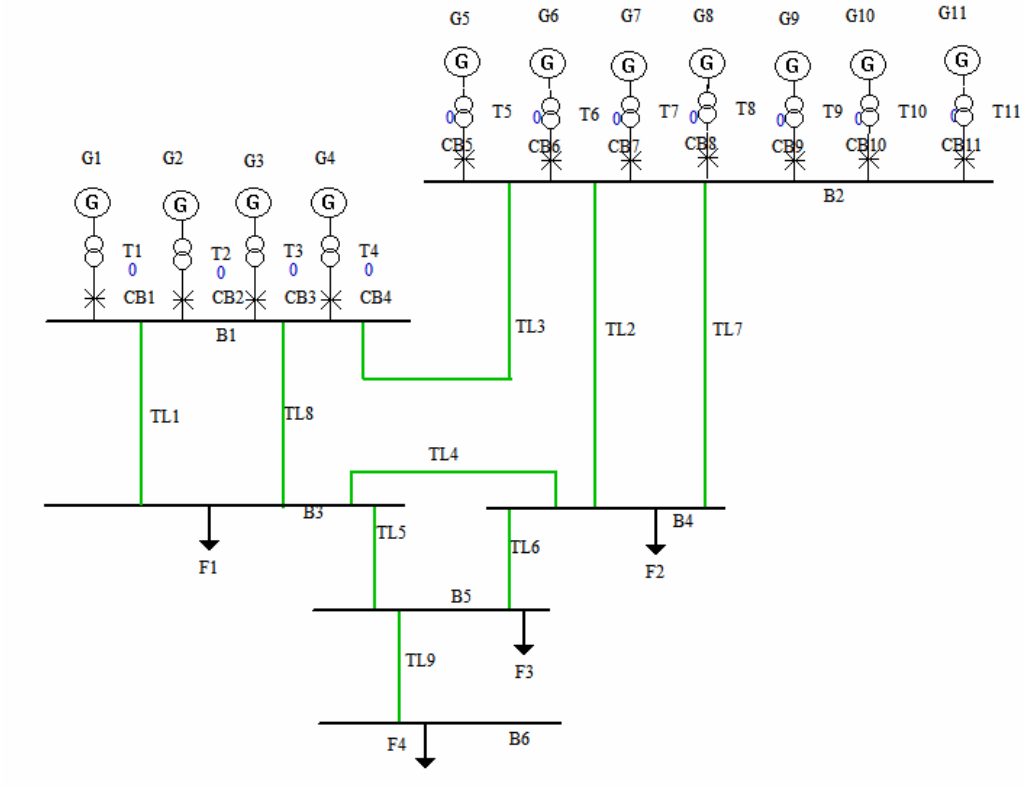


Figure 6-3 RBTS Schematic Diagram

6.4 Node to Node Detection

All the connections between each component must be properly connected for the computation of the phase angle results. The schematic check is done after the transmission network is drawn as shown in Figure 6-4.

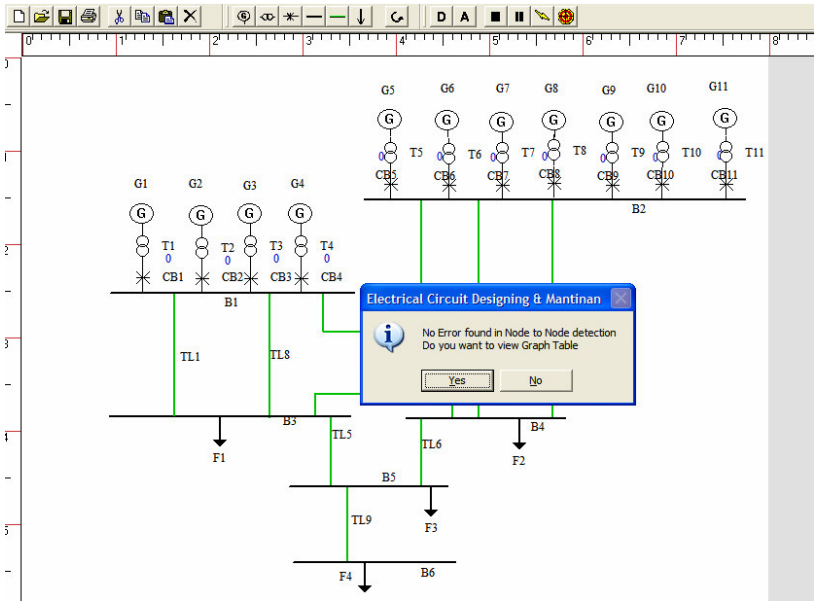


Figure 6-4 Node to Node Detection

A table will appear indicating that there is no error for the schematic diagram, and an option of viewing the transition graph table is also given.

In order to compute the phase angle results for the schematic diagram, the connections between all components must be checked. The results for the connections for all components are shown in the transition graph table. It will show all the components for the schematic diagram and the To-From connection results as shown in Figure 6-5.

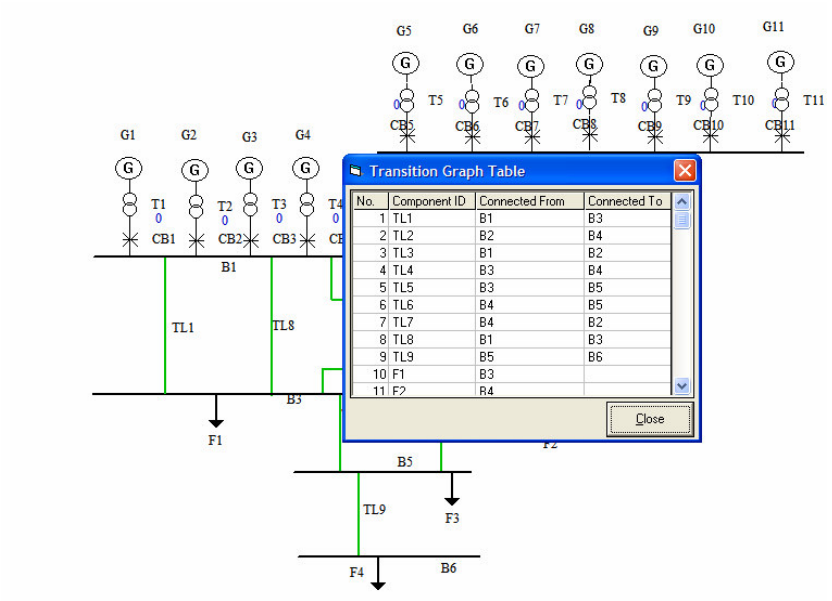


Figure 6-5 Transition Graph Table

For example for Component ID TL3, the transition graph table in Figure 6-5 shows that it is connected from Busbar1 to Busbar2.

6.5 Simulation

6.5.1 Voltage Angles for Each Bus

The schematic diagram must be well connected by using schematic check for further simulation results. After the user keys in the entire component's input data, the voltage angles for busbars are simulated as shown in Figure 6-6.

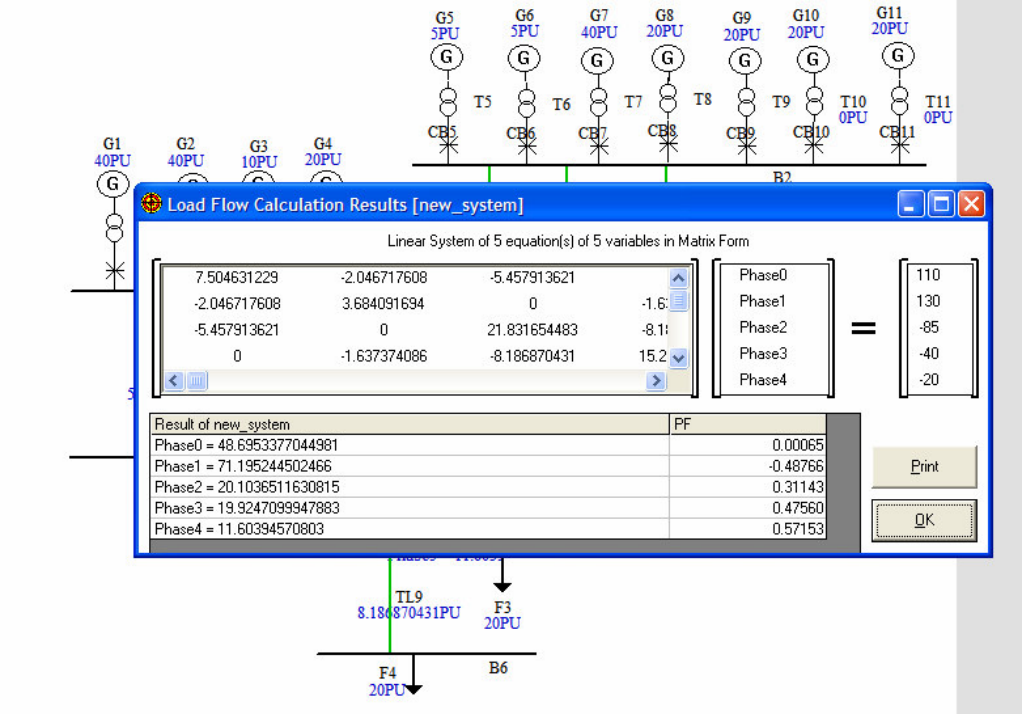


Figure 6-6 Voltage Angles

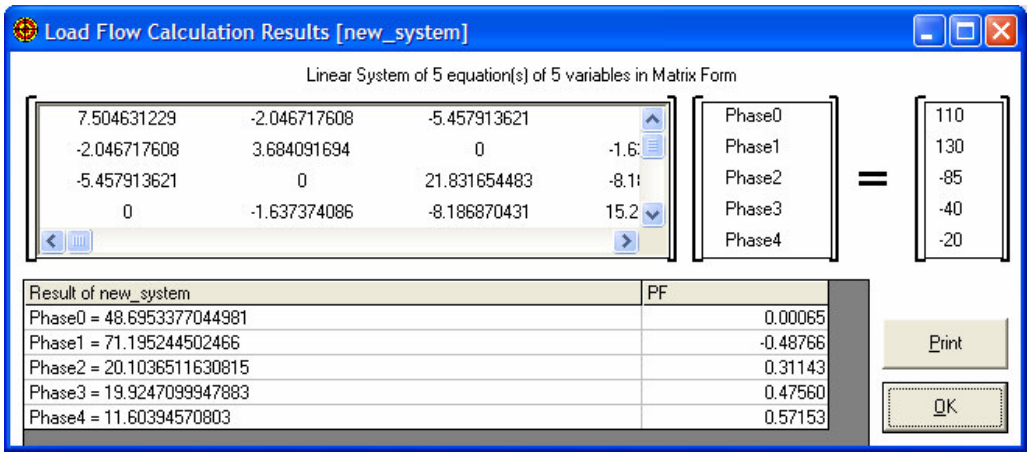


Figure 6-7 Voltage Angles Table

The results of the phase angle and the power factor will be shown on the Load Flow calculation result table in Figure 6-7. This schematic diagram consists of 6 busbars thus the Y matrix that is simulated for the voltage angle is a 5x5 matrix.

6.5.2 Load Flow Direction

After all the voltage angles have been found, the load flow can be shown by the animated direction as shown in Figure 6-8 by pressing the AC/DC button.

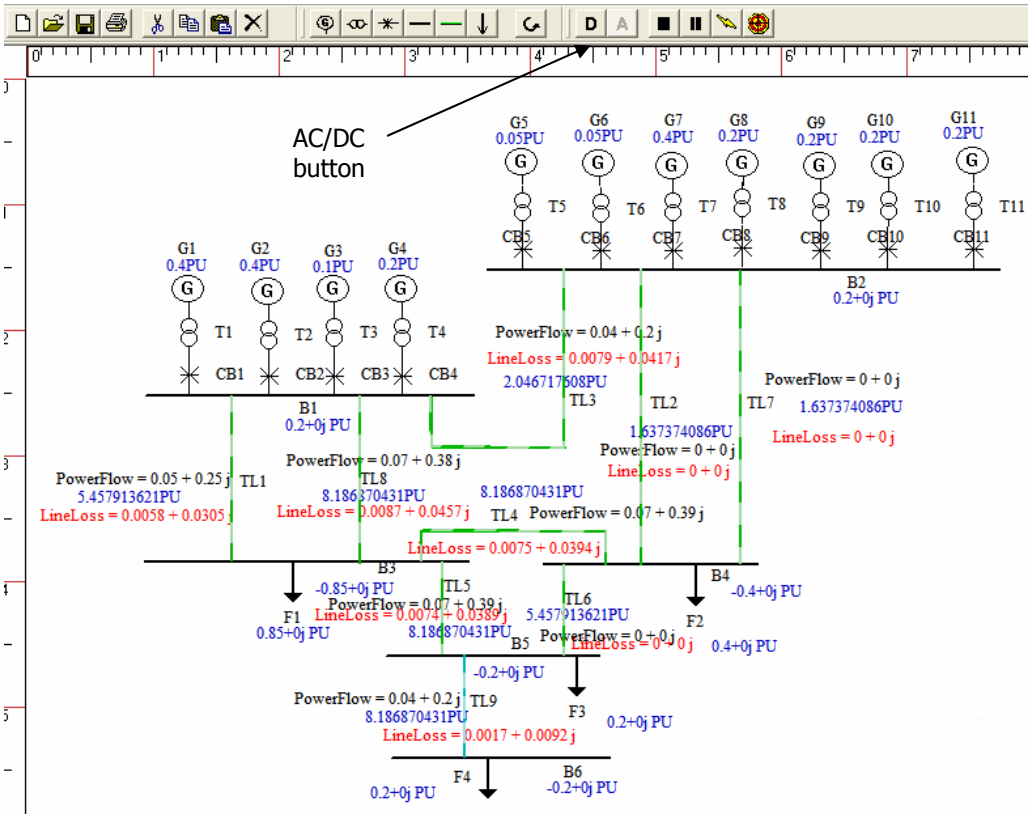


Figure 6-8 Load flow direction

The direction of load flow for TL1 is from busbar1 to busbar3.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

Load flow studies are essential in planning the future development of the system because satisfactory operation of the system depends on knowing the effects of interconnections with other power systems, of new loads, new generating stations, and new transmission lines before they are installed.

This project focuses on the study of load flow analysis as well as the solution techniques on how load flow is calculated. The concepts of failure mode analysis and the use of approximate model of the power system are used to provide an adequate capability.

The developed software is essential as it has the flexibility of design and analysis of electrical power systems. It provides powerful drawing tools that can quickly create a structured schematic diagram. The simulation program is used to check for valid system connections and a database is created to store component data. The load flow direction is shown by using animation.

The benefit for having this software is that multiple one-line diagrams can be associated with each project for better system organization and presentation. With this software, documents that exist in the power systems can be extracted. Evaluation of the Load Flow can be conducted. This has helped the engineers to troubleshoot and analyze a power system network in a more efficient way.

7.2 Recommendation

Every program has its own merit points but it can still be further enhanced to make the program more complete and also to accommodate more users. Some recommendations for our program are listed below:

1. Currently the program for database is a standalone system. It can be built on to become web based so that the users will be able to access it through internet.

2. The AC analysis has a few pre-default values like the BASEMVA, transformer tap setting. It can be improved by adding all these default item in as user input. And also at the same time, use Newton Raphson method to calculate the load flow.

3. The program can also be further enhanced to include other power system calculation. Some examples are: Transient Analysis of Transmission Lines, Harmonics Considerations in the event of a fault and so on.

4. The program help button has already move user to a dummy help file. It may be helpful if more time is available to update all the information in the help text. This help files will be able to provide user a better understanding on how to use our program effectively and efficiently.

5. Currently there are only limited components; it is recommended that more components will be added into the software.

6. The load flow animation for AC analysis do not have the capacity limit as the DC Analysis, it might be good if there is a capacity limit in AC analysis too.

References

1. R. Allan, R. Billinton, S.Kumar, N. Chowdhurry, K.Chu, K. Debnath, L.Goel, E. Khan, P. Kos, G. Noutbakhsh, J. Oteng-Adjei, August 1989, Power Systems Research Group, University of Saskatchewan, Canada. 'IEEE Transactions on Power Systems', Vol 4, No 3,

2. Saadat Hadi, c1999, *Power System Analysis*, WCB/McGraw-Hill, Boston.

3. Grainger John J., Stevenson William D. Jr., c1994, *Power System Analysis*, McGraw-Hill, New York.

4. Gross Charles A., c1986, *Power System Analysis*, Wiley, New York.

5. Glover J. Duncan, c2002, *Power System Analysis and Design*, Wadsworth/Thomson Learning, Pacific Grove, CA.

6. Del Toro Vincent, c1992, *Electric Power System*, Prentice Hall, Englewood Cliffs, N. J..

7. Deitel Harvey M., c2002, *Visual Basic .Net: How To Program*, Prentice Hall ,Upper Saddle River, N. J..

8. Macfadden Fred R., Hoffer Jeffrey A., Prescott Mary B., c1999, *Modern Database Management*, Reading, Mass., Addison-Wesley.

9. Date C. J., 2004, *An Introduction To Database Systems*, Pearson/Addison Wesley, Boston.

10. Gilberto E. Urroz, Civil and Environmental Engineering, College of Engineering, Utah State University, viewed 12th May 2005, http://www.engineering.usu.edu/cee/faculty/gurro/VBA&Excel.htm

11. Exhedra Solutions, Inc., viewed 1st July 2005, http://www.planetsourcecode.com/vb/default.asp?lngWId=1

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

FOR:	**ONG Gimhua**
TOPIC:	SOFTWARE DEVELOPMENT FOR POWER SYSTEM ANALYSIS
SUPERVISIORS:	Ron Sharma
ENROLMENT:	ENG 4111 – S1, EXT, 2005; ENG 4112 – S2, EXT, 2005
PROJECT AIM:	This project aim to develop a software to determine the response of a power supply system in Singapore when the system is subjected to disturbances. The study of dynamic response to sudden changes and disturbances in electrical power systems is carried out. Common disturbances include fault, power flow, switching, load changes, motor starting, loss of generation, loss of excitation, and blocked governor, etc

PROGRAMME: **Issue A, 21st Mar 2005**

8. Create Tool box for drawing.
9. Schematic check program to check all the interconnections of a busbar with other components is performed to form the NxN system.
10. Design a simulation program to simulate power flow.
11. Draw Animation to show load flow direction.
12. Create a database to store component data and system.
13. Perform DC/AC load analysis.
14. The software intended to use for the above includes Microsoft Visual Basic 6 programming language, Microsoft Access 2000 and Matlab simulation.

As time permits:

15. To be built on to web-based so that the users will be able to access it through internet.

AGREED:

_____	_____
Student	Supervisor
ONG Gimhua	Ron Sharma
18 March 2005	_____ 2005

Appendix B

Software Code

B.1 Forms

B.1.1 FrmAdd

```
Public dbpath As String 'for global declaration for database linkage

Private Sub CancelButton_Click()
   Unload Me
End Sub

Private Sub DeleteButton_Click()
Dim db1 As Database
Dim record1 As Recordset

Dim dbpath As String
Dim sPassword As String
Dim SQL1 As String

Dim temp1 As String
Dim counter As Integer
Dim response As Integer

counter = 0

dbpath = "\db1.mdb"
Set db1 = OpenDatabase(App.Path & dbpath, dbDriverComplete, False,
";DATABASE=;VID=;PWD=" & sPassword & ";DSN=")
SQL1 = "SELECT * FROM userdata"

Set record1 = db1.OpenRecordset(SQL1, dbOpenDynaset)

With record1
Do While Not record1.EOF

If textUserID1.Text = !UserID Then
   response = MsgBox("Are You Sure You Want To Delete This Record?", vbYesNoCancel +
vbCritical + vbDefaultButton1, "Delete Record")

If response = vbYes Then
   .Delete
End If

End If
record1.MoveNext
Loop
End With
MsgBox "You have successfully delete a User.", vbInformation
textUserID1.Text = ""
record1.Close
End Sub

Private Sub EnterButton_Click()

Dim db1 As Database
Dim record1 As Recordset
Dim record2 As Recordset

Dim dbpath As String
```

```
Dim sPassword As String
Dim SQL1 As String
Dim SQL2 As String
Dim temp1 As String
Dim counter As Integer
Dim response As Integer

counter = 0

If textUserID.Text = "" Or textPassword.Text = "" Or PrivilegeCombo.Text = "" Or
textConfirm.Text = "" Then
MsgBox "Please fill in  All 4 FIELDS: UserID, Password and Aaccess level respectively !",
vbInformation
Exit Sub
End If

dbpath = "\db1.mdb"
Set db1 = OpenDatabase(App.Path & dbpath, dbDriverComplete, False,
";DATABASE=;VID=;PWD=" & sPassword & ";DSN=")
SQL1 = "SELECT * FROM userdata"
SQL2 = "SELECT * FROM userdata"

Set record1 = db1.OpenRecordset(SQL1, dbOpenDynaset)
Set record2 = db1.OpenRecordset(SQL2, dbOpenDynaset)

With record1
Do While Not record1.EOF

If !UserID = textUserID.Text Or UCase(!UserID) = textUserID.Text Then
MsgBox "Sorry this UserID has already been used,please choose another UserID. ",
vbInformation
textUserID.Text = ""
textPassword.Text = ""
textConfirm.Text = ""
Exit Sub
End If

.MoveNext
Loop
End With
record1.Close
If textPassword.Text = textConfirm.Text Then
   With record2

   .AddNew
   !UserID = textUserID.Text
   !Password = textPassword.Text
   If PrivilegeCombo.Text = "Level 0" Then
      !Privilege = 0
   Else
      !Privilege = 1
   End If
   .Update

   End With
   MsgBox "You have successfully enter a User", vbInformation
   textUserID.Text = ""
   textPassword.Text = ""
   textConfirm.Text = ""
```

```
    PrivilegeCombo = ""
    record2.Close
Else
    MsgBox "Re-confirm Password again.", vbInformation
    textPassword.Text = ""
    textConfirm.Text = ""
End If

End Sub
Private Sub Form_Load()

dbpath = "db1.mdb"  'stores the databse information, such as directory
global_clock = 30000 'default is 30s, else there will be problems entering the figure to the interval property of the timer
global_clock_count = 60 ' such that the total timer is =30min
global_clock_count = global_clock_count - 1 'for proper count of the timer

Frame2.Enabled = False
Frame3.Enabled = False
Option1.Value = False
Option2.Value = False

End Sub

Private Sub Option1_Click()
Frame2.Enabled = True
Frame3.Enabled = False


End Sub

Private Sub Option2_Click()
Frame3.Enabled = True
Frame2.Enabled = False
End Sub

'''additionl
Private Sub Clearing_Data()
'just clears the textfield only
textUserID.Text = ""
textPassword.Text = ""

End Sub
```

B.1.2 FrmChangePW

```
Private Sub CancelButton_Click()
    Unload Me
End Sub

Private Sub ChangeButton_Click()
Dim db1 As Database
Dim record1 As Recordset
Dim record2 As Recordset

Dim dbpath As String
Dim sPassword As String
Dim SQL1 As String
```

```
Dim SQL2 As String
Dim temp1 As String
Dim counter As Integer
Dim response As Integer

counter = 0

'"If textUserID.Text = "" Or textOldPassword.Text = "" Or textNewPassword.Text = "" Or
textConfirm.Text = "" Then
'"MsgBox "Please fill in  All 4 FIELDS: UserID, Old Password, New Password and Confirm
Password !", vbInformation
'"End If

dbpath = "\db1.mdb"
Set db1 = OpenDatabase(App.Path & dbpath, dbDriverComplete, False,
";DATABASE=;VID=;PWD=" & sPassword & ";DSN=")
SQL1 = "SELECT * FROM userdata"
SQL2 = "SELECT * FROM userdata where UserID = '" + textUserID.Text + "'"

Set record1 = db1.OpenRecordset(SQL1, dbOpenDynaset)
Set record2 = db1.OpenRecordset(SQL2, dbOpenDynaset)

If textUserID.Text = "" Or textOldPassword.Text = "" Or textNewPassword.Text = "" Or
textConfirm.Text = "" Then
MsgBox "Please fill in  All 4 FIELDS: UserID, Old Password, New Password and Confirm
Password !", vbInformation
Else

With record1
Do While Not record1.EOF
If !UserID = textUserID.Text Or UCase(!UserID) = textUserID.Text Then
   If !Password = textOldPassword.Text Then
      If textNewPassword.Text = textConfirm.Text Then
         With record2

         .Edit
         !Password = textNewPassword.Text
         .Update

         End With
         MsgBox "You have successfully change the password.", vbInformation
         textUserID.Text = ""
         textOldPassword.Text = ""
         textNewPassword.Text = ""
         textConfirm.Text = ""
         record2.Close
      Else
         MsgBox "Re-confirm Password again.", vbInformation
         textNewPassword.Text = ""
         textConfirm.Text = ""
      End If
   Else
      MsgBox "Invalid Old Password.", vbInformation
      textOldPassword.Text = ""
      textNewPassword.Text = ""
      textConfirm.Text = ""
   End If
Else
   counter = counter + 1
```

```
'Exit Sub
End If
.MoveNext
Loop
End With
'MsgBox "counter " & counter
'MsgBox "record " & record1.RecordCount
If counter = record1.RecordCount Then
   MsgBox "Invalid UserID.", vbCritical
   textUserID.Text = ""
   textOldPassword.Text = ""
   textNewPassword.Text = ""
   textConfirm.Text = ""
End If
record1.Close
End If
End Sub
```

B.1.3 FrmCircuitbreakerFailure

```
'=============================================================
====================================
'=============================================================
====================================
'   THE GOALS OF THIS FORM ARE
'   1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF CircuitBreakerS' FAILURE
'   2.  SHOW SPECIFIC RECORDS RELATED TO A CircuitBreaker OR ALL
'=============================================================
====================================
'=============================================================
====================================


Private Sub CmdClose_Click()
   'JUST HIDING TO CLOSE
   Hide

End Sub

Private Sub CmdMoveFirst_Click()
   'MOVING FIRST ACCORDING TO THE USER CHOICE
   If optAll Then  'VIEW ALL RECORD OF Circuit Breakers
      rsForAll.MoveFirst
   Else 'VIEW ONLY RECORDS WITH SPECIFIC CircuitBreakersID
      'THIS ID IS EQUAL txtCircuitBreakersID.text
      rsForAll.FindFirst "CircuitBreakersID='" & txtCircuitBreakersID & "'"
   End If
   LoadFormData Me 'LOADING DATA TO THE CURRENT FORM

End Sub

Private Sub CmdMoveLast_Click()
   'MOVING LAST ACCORDING TO THE USER CHOICE
   If optAll Then
      rsForAll.MoveLast
   Else 'VIEW ONLY RECORDS WITH SPECIFIC CircuitBreakersID
      'THIS ID IS EQUAL txtCircuitBreakersID.text
```

```
        rsForAll.FindLast "CircuitBreakersID='" & txtCircuitBreakersID & "'"
    End If
    LoadFormData Me 'LOADING DATA TO THE CURRENT FORM
End Sub

Private Sub CmdMoveNext_Click()
    'MOVING NEXT ACCORDING TO THE USER CHOICE
    If optAll Then
        rsForAll.MoveNext
        If rsForAll.EOF Then
            MsgBox "This is Last record", vbInformation
            rsForAll.MoveLast
        End If
    Else 'VIEW ONLY RECORDS WITH SPECIFIC CircuitBreakersID
        'THIS ID IS EQUAL txtCircuitBreakersID.text
        rsForAll.FindNext "CircuitBreakersID='" & txtCircuitBreakersID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is Last record", vbInformation
        End If
    End If
    LoadFormData Me 'LOADING DATA TO THE CURRENT FORM

End Sub

Private Sub CmdMovePrevious_Click()
    'MOVING PREVIOUS ACCORDING TO THE USER CHOICE
    If optAll Then
        rsForAll.MovePrevious
        If rsForAll.BOF Then
            MsgBox "This is first record", vbInformation
            rsForAll.MoveLast
        End If
    Else 'VIEW ONLY RECORDS WITH SPECIFIC CircuitBreakersID
        'THIS ID IS EQUAL txtCircuitBreakersID.text
         rsForAll.FindPrevious "CircuitBreakersID='" & txtCircuitBreakersID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is first record", vbInformation
        End If
    End If
    LoadFormData Me 'LOADING DATA TO THE CURRENT FORM

End Sub

Private Sub CmdUpdate_Click()

    SaveFormData Me, UpdateRecord   'UPDATING RECORD OF THIS FORM
    MsgBox "Record is updated", vbOKOnly + vbInformation, "Record Updation"

End Sub

Private Sub CmdShowDate_Click()

    txtFailureDate = GetDate(Me) 'GETING DATE BY DISPLAYING A CALENDER

End Sub
```

B.1.4 FrmCircuitbreakers

```
'=========================================================
==================================
'=========================================================
==================================
'  THE GOALS OF THIS FORM IS
'   1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF CircuitBreakerS' BASIC
PARAMETERS
'=========================================================
==================================
'=========================================================
==================================
Option Explicit
Private MyMaitinadate As New LocalDate, ChangeMDate As Boolean

Private Sub Form_Activate()

   If txtDateInstalled = "__/__/____" Or txtDateInstalled = "" Then
      txtDateInstalled = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") & "/"
& Year(Date)
      MyMaitinadate.Value = txtDateInstalled
   Else
      MyMaitinadate.Value = txtNextMaintenanceDate
   End If
   ChangeMDate = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ChangeMDate = False
End Sub

Private Sub txtDateInstalled_Change()

   'The following change will take place when
   'InstalledDate is not empty AND current form is displayed on screen
   If txtDateInstalled <> "__/__/____" And ChangeMDate Then
      MyMaitinadate.Value = txtDateInstalled
      Call txtMaintenanceInterval_Change
   End If

End Sub

Private Sub txtMaintenanceInterval_Change()

   'On Error Resume Next
   If ChangeMDate Then txtNextMaintenanceDate =
MyMaitinadate.AddDays(CLng(Val(txtMaintenanceInterval)))

End Sub

Private Sub cmdCancel_Click()

   Hide
   ChangeMDate = False

End Sub

Private Sub cmdSAVE_Click()
```

```
    SaveFormData Me, UpdateRecord
    Hide
    ChangeMDate = False
End Sub

Private Sub CmdShowDate_Click()
    Dim Temp As String
    Temp = GetDate(Me)
    If Temp <> "" Then txtDateInstalled = Temp 'GETING DATE BY DISPLAYING A
CALENDER

End Sub

Private Sub CmdShowDate2_Click()
    Dim Temp As String
    Temp = GetDate(Me)
    If Temp <> "" Then txtNextMaintenanceDate = Temp          'GETING DATE BY
DISPLAYING A CALENDER

End Sub
```

B.1.5 FrmDimensions

```
Public ApplyChange As Boolean

Private Sub cmdCancel_Click()
    ApplyChange = False
    Hide
End Sub

Private Sub cmdOK_Click()
    ApplyChange = True
    Hide
End Sub

Private Sub Form_Load()
    For I = 4 To 24
        ComboWidth.AddItem Format(I, "0.0")
        ComboHeight.AddItem Format(I, "0.0")
        For J = 1 To 9
            ComboWidth.AddItem Format(I + J * 0.1, "0.0")
            ComboHeight.AddItem Format(I + J * 0.1, "0.0")
        Next
    Next
    ComboWidth.AddItem "25.0"
    ComboHeight.AddItem "25.0"

End Sub
```

B.1.6 FrmFeeder

```
'=========================================================
==================================
'=========================================================
==================================
'  THE GOALS OF THIS FORM IS
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF FEEDERS' BASIC
PARAMETERS
'=========================================================
==================================
'=========================================================
==================================
'THIS FORM IS SIMILAR TO "FrmCircuitBreakers"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitBreakers"

Private Sub cmdCancel_Click()

    Me.Hide

End Sub

Private Sub cmdSAVE_Click()

    SaveFormData Me, UpdateRecord
    Hide

End Sub

Private Sub CmdShowDate_Click()

    txtDateInstalled = GetDate(Me)

End Sub

Private Sub CmdShowDate2_Click()

    txtNextMaintenanceDate = GetDate(Me)

End Sub
```

B.1.7 FrmGenerator

```
'=========================================================
==================================
'=========================================================
==================================
'  THE GOALS OF THIS FORM IS
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF GENERATR'S BASIC
PARAMETERS
'=========================================================
==================================
'=========================================================
==================================
'THIS FORM IS SIMILAR TO "FrmCircuitBreakers"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitBreakers"
Option Explicit
```

```
Private MyMaitinadate As New LocalDate, ChangeMDate As Boolean

Private Sub Form_Activate()

   If txtDateInstalled = "__/__/____" Or txtDateInstalled = "" Then
      txtDateInstalled = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") & "/"
& Year(Date)
      MyMaitinadate.Value = txtDateInstalled
   Else
      MyMaitinadate.Value = txtNextMaintenanceDate
   End If
   ChangeMDate = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ChangeMDate = False
End Sub


Private Sub txtDateInstalled_Change()

   'The following change will take place when
   'InstalledDate is not empty AND current form is displayed on screen
   If txtDateInstalled <> "__/__/____" And ChangeMDate Then
      MyMaitinadate.Value = txtDateInstalled
      Call txtMaintenanceInterval_Change
   End If

End Sub

Private Sub txtMaintenanceInterval_Change()

   'On Error Resume Next
   If ChangeMDate Then txtNextMaintenanceDate =
MyMaitinadate.AddDays(CLng(Val(txtMaintenanceInterval)))

End Sub

Private Sub cmdCancel_Click()

   Hide
   ChangeMDate = False

End Sub

Private Sub cmdSAVE_Click()
   SaveFormData Me, UpdateRecord
   Hide
   ChangeMDate = False
End Sub

Private Sub CmdShowDate_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtDateInstalled = Temp 'GETING DATE BY DISPLAYING A
CALENDER

End Sub
```

```
Private Sub CmdShowDate2_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtNextMaintenanceDate = Temp          'GETING DATE BY
DISPLAYING A CALENDER

End Sub
```

B.1.8 FrmGeneratorFailure

```
'=============================================================
====================================
'=============================================================
====================================
'   THE GOALS OF THIS FORM ARE
'   1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF GENERATORS' FAILURE
'   2.  SHOW SPECIFIC RECORDS RELATED TO A GENERATOR OR ALL
'=============================================================
====================================
'=============================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitbreakerFailure"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitbreakerFailure"

Private Sub CmdClose_Click()

   Hide

End Sub

Private Sub CmdMoveFirst_Click()

   If optAll Then
      rsForAll.MoveFirst
   Else
      rsForAll.FindFirst "generatorsid='" & txtGeneratorsID & "'"
   End If
   LoadFormData Me

End Sub

Private Sub CmdMoveLast_Click()
   If optAll Then
      rsForAll.MoveLast
   Else
      rsForAll.FindLast "generatorsid='" & txtGeneratorsID & "'"
   End If
   LoadFormData Me
End Sub

Private Sub CmdMoveNext_Click()

   If optAll Then
      rsForAll.MoveNext
      If rsForAll.EOF Then
         MsgBox "This is Last record", vbInformation
         rsForAll.MoveLast
      End If
```

```
    Else
        rsForAll.FindNext "generatorsid='" & txtGeneratorsID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is Last record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdMovePrevious_Click()

    If optAll Then
        rsForAll.MovePrevious
        If rsForAll.BOF Then
            MsgBox "This is first record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindPrevious "generatorsid='" & txtGeneratorsID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is first record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdUpdate_Click()

    SaveFormData Me, UpdateRecord
    MsgBox "Record is updated", vbOKOnly + vbInformation, "Record Updation"

End Sub

Private Sub CmdShowDate_Click()

    txtFailureDate = GetDate(Me)

End Sub
```

B.1.9 FrmGetDate

```
Public IsDateSelected  As Boolean

Private Sub cmdCancel_Click()

    IsDateSelected = False
    Me.Hide

End Sub

Private Sub cmdOK_Click()

    IsDateSelected = True
    Me.Hide

End Sub
```

B.1.10 FrmGraphTable

```
Public Sub Clear()

   grdGraph.Rows = 1
   grdGraph.Cols = 4
   grdGraph.ColWidth(0) = 500
   grdGraph.ColWidth(1) = 1200
   grdGraph.ColWidth(2) = 1500
   grdGraph.ColWidth(3) = 1200
   grdGraph.Width = 500 + 1200 + 1500 + 1500 + 60
   Width = grdGraph.Width + 200
   CmdClose.Left = grdGraph.Left + grdGraph.Width - CmdClose.Width
   grdGraph.TextMatrix(0, 0) = "No."
   grdGraph.TextMatrix(0, 1) = "Component ID"
   grdGraph.TextMatrix(0, 2) = "Connected From"
   grdGraph.TextMatrix(0, 3) = "Connected To"

End Sub

Public Sub Add(ComponentID As String)

   grdGraph.Rows = grdGraph.Rows + 1
   grdGraph.TextMatrix(grdGraph.Rows - 1, 0) = grdGraph.Rows - 1
   grdGraph.Row = grdGraph.Rows - 1
   grdGraph.col = 1
   grdGraph.Text = ComponentID

End Sub

Public Sub AddConnectedTo(connectedto As String)

   grdGraph.col = 3
   grdGraph.Text = connectedto

End Sub

Public Sub AddConnectedFrom(ConnectedFrom As String)

   grdGraph.col = 2
   grdGraph.Text = ConnectedFrom

End Sub

Private Sub CmdClose_Click()

   Hide

End Sub
```

B.1.11 FrmLineBusbar

```
'=========================================================
====================================
'=========================================================
====================================
'  THE GOALS OF THIS FORM IS
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF BUSBAR'S BASIC
PARAMETERS
'=========================================================
====================================
'=========================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitBreakers"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitBreakers"
Option Explicit
Private MyMaitinadate As New LocalDate, ChangeMDate As Boolean

Private Sub Form_Activate()

   If txtDateInstalled = "__/__/____" Or txtDateInstalled = "" Then
      txtDateInstalled = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") & "/"
& Year(Date)
      MyMaitinadate.Value = txtDateInstalled
   Else
      MyMaitinadate.Value = txtNextMaintenanceDate
   End If
   ChangeMDate = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ChangeMDate = False
End Sub

Private Sub txtDateInstalled_Change()

   'The following change will take place when
   'InstalledDate is not empty AND current form is displayed on screen
   If txtDateInstalled <> "__/__/____" And ChangeMDate Then
      MyMaitinadate.Value = txtDateInstalled
      Call txtMaintenanceInterval_Change
   End If

End Sub

Private Sub txtMaintenanceInterval_Change()

   'On Error Resume Next
   If ChangeMDate Then txtNextMaintenanceDate =
MyMaitinadate.AddDays(CLng(Val(txtMaintenanceInterval)))

End Sub

Private Sub cmdCancel_Click()

   Hide
   ChangeMDate = False

End Sub
```

```
Private Sub cmdSAVE_Click()

   SaveFormData Me, UpdateRecord
   Hide
   ChangeMDate = False
End Sub

Private Sub CmdShowDate_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtDateInstalled = Temp 'GETING DATE BY DISPLAYING A
CALENDER

End Sub

Private Sub CmdShowDate2_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtNextMaintenanceDate = Temp          'GETING DATE BY
DISPLAYING A CALENDER

End Sub
```

B.1.12 FrmLineBusbarFailure

```
'=============================================================
=====================================
'=============================================================
=====================================
'   THE GOALS OF THIS FORM ARE
'   1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF BUSBARS' FAILURE
'   2.  SHOW SPECIFIC RECORDS RELATED TO A BUBAR OR ALL
'=============================================================
=====================================
'=============================================================
=====================================
'THIS FORM IS SIMILAR TO "FrmCircuitbreakerFailure"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitbreakerFailure"
Private Sub CmdClose_Click()

   Hide

End Sub

Private Sub CmdMoveFirst_Click()

   If optAll Then
      rsForAll.MoveFirst
   Else
      rsForAll.FindFirst "busbarid='" & txtBusbarID & "'"
   End If
   LoadFormData Me

End Sub

Private Sub CmdMoveLast_Click()
   If optAll Then
      rsForAll.MoveLast
```

```
    Else
        rsForAll.FindLast "busbarid='" & txtBusbarID & "'"
    End If
    LoadFormData Me
End Sub

Private Sub CmdMoveNext_Click()

    If optAll Then
        rsForAll.MoveNext
        If rsForAll.EOF Then
            MsgBox "This is Last record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindNext "busbarid='" & txtBusbarID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is Last record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdMovePrevious_Click()

    If optAll Then
        rsForAll.MovePrevious
        If rsForAll.BOF Then
            MsgBox "This is first record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindPrevious "busbarid='" & txtBusbarID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is first record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdUpdate_Click()

    SaveFormData Me, UpdateRecord
    MsgBox "Record is updated", vbOKOnly + vbInformation, "Record Updation"

End Sub

Private Sub CmdShowDate_Click()

    txtFailureDate = GetDate(Me)

End Sub
```

B.1.13 FrmLineTransmission

```
'=========================================================
====================================
'=========================================================
====================================
'   THE GOALS OF THIS FORM IS
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF TRANSMISSION LINE'S
BASIC PARAMETERS
'=========================================================
====================================
'=========================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitBreakers"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitBreakers"
Option Explicit
Private MyMaitinadate As New LocalDate, ChangeMDate As Boolean

Private Sub Form_Activate()

   If txtDateInstalled = "__/__/____" Or txtDateInstalled = "" Then
      txtDateInstalled = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") & "/"
& Year(Date)
      MyMaitinadate.Value = txtDateInstalled
   Else
      MyMaitinadate.Value = txtNextMaintenanceDate
   End If
   ChangeMDate = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ChangeMDate = False
End Sub


Private Sub txtCapacitancePerKm_Change()

   If Me.txtCapacitancePerKm = "" And Me.txtInductancePerKm = "" And
Me.txtResistancePerKm = "" And Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = 0
      Me.txtTotalCapacitance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtCapacitancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalCapacitance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtCapacitancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = 0
      Me.txtTotalCapacitance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
```

```
        Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
        Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = 0
    ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm = "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
        Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
        Me.txtTotalResistance = 0
    ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm = "" And Me.txtTotalLength <> "" Then
        Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
        Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
        Me.txtTotalCapacitance = 0
    ElseIf Me.txtTotalLength <> "" Then
        Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
        Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
    End If

End Sub

Private Sub txtDateInstalled_Change()

    'The following change will take place when
    'InstalledDate is not empty AND current form is displayed on screen
    If txtDateInstalled <> "__/__/____" And ChangeMDate Then
        MyMaitinadate.Value = txtDateInstalled
        Call txtMaintenanceInterval_Change
    End If

End Sub

Private Sub txtInductancePerKm_Change()

    If Me.txtCapacitancePerKm = "" And Me.txtInductancePerKm = "" And
Me.txtResistancePerKm = "" And Me.txtTotalLength <> "" Then
        Me.txtTotalInductance = 0
        Me.txtTotalCapacitance = 0
        Me.txtTotalResistance = 0
    ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
        Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = 0
        Me.txtTotalResistance = 0
    ElseIf Me.txtCapacitancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
        Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
        Me.txtTotalCapacitance = 0
        Me.txtTotalResistance = 0
    ElseIf Me.txtInductancePerKm = "" And Me.txtCapacitancePerKm = "" And
Me.txtTotalLength <> "" Then
        Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = 0
        Me.txtTotalCapacitance = 0
    ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
        Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
        Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
        Me.txtTotalInductance = 0
```

```
   ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm = "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm = "" And Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
      Me.txtTotalCapacitance = 0
   ElseIf Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
   End If

End Sub

Private Sub txtMaintenanceInterval_Change()

   'On Error Resume Next
   If ChangeMDate Then txtNextMaintenanceDate =
MyMaitinadate.AddDays(CLng(Val(txtMaintenanceInterval)))

End Sub

Private Sub cmdCancel_Click()

   Hide
   ChangeMDate = False

End Sub

Private Sub cmdSAVE_Click()

   SaveFormData Me, UpdateRecord
   Hide
   ChangeMDate = False
End Sub

Private Sub CmdShowDate_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtDateInstalled = Temp 'GETING DATE BY DISPLAYING A
CALENDER

End Sub

Private Sub CmdShowDate2_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtNextMaintenanceDate = Temp 'GETING DATE BY
DISPLAYING A CALENDER

End Sub

Private Sub txtResistancePerKm_Change()
```

```
   If Me.txtCapacitancePerKm = "" And Me.txtInductancePerKm = "" And
Me.txtResistancePerKm = "" And Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = 0
      Me.txtTotalCapacitance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtCapacitancePerKm = "" And Me.txtResistancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalCapacitance = 0
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtCapacitancePerKm = "" And
Me.txtTotalLength <> "" Then
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = 0
      Me.txtTotalCapacitance = 0
   ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = 0
   ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm = "" And
Me.txtCapacitancePerKm <> "" And Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = 0
   ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm = "" And Me.txtTotalLength <> "" Then
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
      Me.txtTotalCapacitance = 0
   ElseIf Me.txtTotalLength <> "" Then
      Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
      Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
      Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
   End If

End Sub

Private Sub txtTotalLength_LostFocus()

   If Me.txtTotalLength = "" Then
      MsgBox "Please Key in the length of the line"
      Me.txtTotalLength.SetFocus
   Else
      If Me.txtCapacitancePerKm = "" And Me.txtInductancePerKm = "" And
Me.txtResistancePerKm = "" Then
         Me.txtTotalInductance = 0
         Me.txtTotalCapacitance = 0
         Me.txtTotalResistance = 0
      ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm = "" Then
         Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
         Me.txtTotalInductance = 0
         Me.txtTotalResistance = 0
      ElseIf Me.txtCapacitancePerKm = "" And Me.txtResistancePerKm = "" Then
```

```
            Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
            Me.txtTotalCapacitance = 0
            Me.txtTotalResistance = 0
        ElseIf Me.txtInductancePerKm = "" And Me.txtCapacitancePerKm = "" Then
            Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
            Me.txtTotalInductance = 0
            Me.txtTotalCapacitance = 0
        ElseIf Me.txtInductancePerKm = "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm <> "" Then
            Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
            Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
            Me.txtTotalInductance = 0
        ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm = "" And
Me.txtCapacitancePerKm <> "" Then
            Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
            Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
            Me.txtTotalResistance = 0
        ElseIf Me.txtInductancePerKm <> "" And Me.txtResistancePerKm <> "" And
Me.txtCapacitancePerKm = "" Then
            Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
            Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
            Me.txtTotalCapacitance = 0
        Else
            Me.txtTotalCapacitance = Me.txtCapacitancePerKm * Me.txtTotalLength
            Me.txtTotalInductance = Me.txtInductancePerKm * Me.txtTotalLength
            Me.txtTotalResistance = Me.txtResistancePerKm * Me.txtTotalLength
        End If
    End If
End Sub
```

B.1.14 FrmLineTransmissionFailure

```
'=============================================================
====================================
'=============================================================
====================================
'   THE GOALS OF THIS FORM ARE
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF TRANSMISSION LINES'
FAILURE
'   2.  SHOW SPECIFIC RECORDS RELATED TO A TRANSMISSION LINE OR ALL
'=============================================================
====================================
'=============================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitbreakerFailure"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitbreakerFailure"
Private Sub CmdClose_Click()

    Hide

End Sub

Private Sub CmdMoveFirst_Click()

    If optAll Then
        rsForAll.MoveFirst
    Else
```

```
        rsForAll.FindFirst "TransmissionLinesid='" & txtTransmissionLinesID & "'"
    End If
    LoadFormData Me

End Sub

Private Sub CmdMoveLast_Click()
    If optAll Then
        rsForAll.MoveLast
    Else
        rsForAll.FindLast "TransmissionLinesid='" & txtTransmissionLinesID & "'"
    End If
    LoadFormData Me
End Sub

Private Sub CmdMoveNext_Click()

    If optAll Then
        rsForAll.MoveNext
        If rsForAll.EOF Then
            MsgBox "This is Last record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindNext "TransmissionLinesid='" & txtTransmissionLinesID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is Last record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdMovePrevious_Click()

    If optAll Then
        rsForAll.MovePrevious
        If rsForAll.BOF Then
            MsgBox "This is first record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindPrevious "TransmissionLinesid='" & txtTransmissionLinesID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is first record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdUpdate_Click()

    SaveFormData Me, UpdateRecord
    MsgBox "Record is updated", vbOKOnly + vbInformation, "Record Updation"

End Sub

Private Sub CmdShowDate_Click()
```

```
    txtFailureDate = GetDate(Me)

End Sub
```

B.1.15 Frmlogin

```
Public dbpath As String
Public globalusername As String
Public global_clock As Integer
Public global_clock_count As Integer
Public global_clock_clock As Integer

Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As
Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String,
ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
Const HELP_CONTENTS = &H3
Private Declare Function WinHelp Lib "User32" Alias "WinHelpA" (ByVal hWnd As Long,
ByVal lpHelpFile As String, ByVal wCommand As Long, dwData As Long) As Long
Dim Var1 As Integer

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub Cmdhelp_Click()
Dim RVal As Integer
    RVal = WinHelp(help.hWnd, "E:\USQ works\Current Module\4111\My
Project\Run\help.hlp", HELP_CONTENTS, 0)
    'Me.HelpContextID = 0
    'Unload Me
    'Call Load(Form1)
    'Call Form1.Show
    'Shell "notepad.exe c:\testing.txt", 1
    'Call Load(MainWindow) 'at the moment did nothing, just load a form, might try
soemthing on this
    'Call MainWindow.Show
End Sub

Private Sub cmdOK_Click()
'   If Combousername(0).Text = "Manager" Then
'       If Textpassword.Text <> "hello" Then
'           Select Case Var1
'               Case Is <= 2
'                   MsgBox "The password is incorrect", vbExclamation, "Incorrect Password"
'                   Var1 = Var1 + 1
'               Case Else
'                       MsgBox "You have entered the wrong password too many times, you are
now locked out.  Please contact your system administrator.  The program will now
terminate.", vbCritical, "Locked Out"
'                   Unload Me
'           End Select
'       Else
'           Call Load(MainWindow)
'           Call MainWindow.Show
'           Unload Me
'       End If
'   ElseIf Combousername(0).Text = "Technician" Then
```

```
'           If Textpassword.Text <> "hello" Then
'              Select Case Var1
'                 Case Is <= 2
'                    MsgBox "The password is incorrect", vbExclamation, "Incorrect Password"
'                    Var1 = Var1 + 1
'                 Case Else
'                     MsgBox "You have entered the wrong password too many times, you are
now locked out.  Please contact your system administrator.  The program will now
terminate.", vbCritical, "Locked Out"
'                    Unload Me
'              End Select
'           Else
'              Call Load(MainWindow)
'              Call MainWindow.Show
'              Unload Me
'           End If
'       ElseIf Combousername(0).Text = "Engineer" Then
'           If Textpassword.Text <> "hello" Then
'              Select Case Var1
'                 Case Is <= 2
'                    MsgBox "The password is incorrect", vbExclamation, "Incorrect Password"
'                    Var1 = Var1 + 1
'                 Case Else
'                     MsgBox "You have entered the wrong password too many times, you are
now locked out.  Please contact your system administrator.  The program will now
terminate.", vbCritical, "Locked Out"
'                    Unload Me
'              End Select
'           Else
'              Call Load(MainWindow)
'              Call MainWindow.Show
'              Unload Me
'          End If
'    End If

'End Sub

'''Changes here
Dim db10 As Database
Dim record1 As Recordset
Dim wrong As Integer
wrong = 0

'check to ensure that both the Username and password textboxes are filled
If Combousername.Text = "" And Textpassword.Text = "" Then
       MsgBox "Please make sure that you login properly", vbCritical, "Login Error"
'check to ensure that if ever the username textbox is not filled
ElseIf Combousername.Text = "" And Not Textpassword.Text = "" Then
       MsgBox "Please enter your Username", vbExclamation, "Login Error"
       Combousername.SetFocus
'check to ensure that if ever the password textbox is not filled
ElseIf Not Combousername.Text = "" And Textpassword.Text = "" Then
       MsgBox "Please enter your Password", vbExclamation, "Login Error"
       Textpassword.SetFocus
Else

   Set db10 = OpenDatabase(dbpath)
   Set Res = db10.OpenRecordset("userdata")
   'MsgBox "database = " & dbpath
```

```
    For I = 1 To Res.RecordCount
        UserID = Res.Fields("UserID")
        'MsgBox "userid" & I & "=" & userid
        Password = Res.Fields("Password")
        'MsgBox "password" & I & "=" & Password

        If Combousername.Text = UserID And Textpassword.Text = Password Then
            Call Load(MainWindow)
            Call MainWindow.Show
            Unload Me
        ElseIf Combousername.Text = UserID And Textpassword.Text <> Password Then
            MsgBox "Invalid Password", vbInformation
            Textpassword.Text = ""
        ElseIf Combousername.Text <> UserID Then
            wrong = wrong + 1
            'MsgBox "wrong" & wrong
        End If
    Res.MoveNext
    Next
    'incase that the login username and the password fields are not correct
    If wrong = Res.RecordCount Then
    MsgBox "UserID not found", vbCritical, "Login Error"
    Clearing_Data
    Combousername.SetFocus
    End If
End If
'"end here

End Sub 'Function
'"additional
Private Sub Form_Load()

'this is a one time entry, then every page will tapped from this page
dbpath = "db1.mdb"  'stores the databse information, such as directory
global_clock = 30000 'default is 30s, else there will be problems entering the figure to the
interval property of the timer
global_clock_count = 60 ' such that the total timer is =30min
global_clock_count = global_clock_count - 1 'for proper count of the timer

Clearing_Data 'call for a function
Timerlogin.Interval = 1
End Sub
'"additionl
Private Sub Clearing_Data()
'just clears the textfield only
Combousername.Text = ""
Textpassword.Text = ""

End Sub



Private Sub Picturelogo_Click()
ShellExecute hWnd, "open", "http://www.usq.edu.au", vbNullString, vbNullString,
conSwNormal
End Sub

Private Sub RonSharma_Click()
```

```
ShellExecute hWnd, "open",
"http://www.usq.edu.au/faculty/engsurv/staff/StaffDetails.asp?Username=sharma",
vbNullString, vbNullString, conSwNormal
End Sub

Private Sub Textpassword_KeyPress(KeyAscii As Integer)
   If KeyAscii = 13 Then  ' The ENTER key.
      SendKeys "{tab}"    ' Set the focus to the next control.
      KeyAscii = 0        ' Ignore this key.
   End If
End Sub

Private Sub Timerlogin_Timer()
   Labeldatetime = Time & "  " & Date
End Sub
```

B.1.16 FrmMain

```
'Public Function Sqrt(ByVal vX As Variant) As Variant

'End Function

'=============================================================
=====================================
'=============================================================
=====================================
'  THE GOALS OF THIS FORM ARE
'  1.  PROVIDE THE USER A SPACE TO MAKE ELECTRICAL CUIRCUITS
'  2.  CREATING FOLLOWING CONTROLS ON USER REQUEST
'      A.  TRANFORMER
'      B.  GENERATOR
'      C.  BUSBAR LINE
'      D.  TRANSMISSION LINE
'      E.  FEEDER
'  3.  CAPTURE ALL INFORMATION OF ALL CONTROLS BY INOVOKING DIFFERENT FORMS
'  4.  SAVING INFORMATION TEMPRARY IN .BK FILE
'  5.  SAVING CUIRCUIT MODEL FILE WITH ALL OF THE CUIRCUT INFORMATION AS .MDB
'
'  -->>  THIS FORM COMMUNICATE WITH THE MAINWINDOW FORM BY MEAN OF
'         1.  MODULE "modPublicProcedures"
'         2.  INTEGER VARIABLE "MyID"
'         3.  AND SOME PUBLIC VARIABLES DEFINED IN MODULE "modCommonVariables"
'=============================================================
=====================================
'=============================================================
=====================================

Option Explicit

Public DB As Database      'Database to which save the model structure and it's information
Public MyFileName As String 'This is the file name by which are used to save model
      'If it is null, it means not save the file yet. Working on only bk file
Public MyID As Integer ' Hold the id of the current form
'This id is usefull to communicate between this form and module "modPublicProcedures"

Private SaveMe As Boolean       'Weather this Model is saved or not?
Private ldBlinkLAL As New LoadUnload     'This hold the information that, weather to blink a
label or not
```

```
Private BlinkLAL() As Label     'This hold the informations of labels to blink.
        'Only lable BlinkLAL(index) blinks if ldBlinkLAL.IsExist(index)

Private ldLineCable As New LoadUnload              'Store weather a Cable(index) is loaded or
not
Private ldLineBusbar As New LoadUnload             'Store weather a Busbar Line(index) is
loaded or not
Private ldLineTransmission As New LoadUnload       'Store weather a Transmission line(index)
is loaded or not
'Private ldBusbar As New LoadUnload                'Store weather a Busbar is loaded or not
Private ldtransformer As New LoadUnload            'Store weather a Transformer(index) is
loaded or not
Private ldGenerator As New LoadUnload              'Store weather a Generator(index) is loaded
or not
Private ldCircuitBreaker As New LoadUnload         'Store weather a CircuitBearker(index) is
loaded or not
Private ldFeeder As New LoadUnload                 'Store weather a Cable Feeder(index) loaded
or not
'Private ldSource As New LoadUnload                'Store weather a Cable Source(index) loaded
or not

Private EditLineCable As Boolean            'Sotre answer of "Is user editing Cable Line?"
Private EditLineBusbar As Boolean           'Sotre answer of "Is user editing Busbar Line?"
Private EditLineTransmission As Boolean     'Sotre answer of "Is user editing Transmission
Line?"
Private EditBusbar As Boolean               'Sotre answer of "Is user editing Busbar?"
Private EditTransformer As Boolean          'Sotre answer of "Is user editing Transformer?"
Private EditGenerator As Boolean            'Sotre answer of "Is user editing Generator?"
Private EditCuircuitBreaker As Boolean      'Sotre answer of "Is user editing
CuircuitBreaker?"
Private EditFeeder As Boolean               'Sotre answer of "Is user editing Feeder?"
Private EditSource As Boolean               'Sotre answer of "Is user editing Source?"

Private Current As Integer    'It is in fact the index of the currently selected components
Public CurrentControlID As String  'It hold the selected component ID

Private Editing As Boolean      'Used weather user is editing or not
        'One can think that
        'When EditLineBusbar,EditSource etc--> Is the "editing" neccessary?
        'Editing is not neccessary, but it makes the program efficient

Private bTransmissionLineBend1 As New LoadUnload        'Used weather user bending a
transmission line or not
Private bTransmissionLineBend2 As New LoadUnload        'Used weather user bending a
transmission line or not
Private MyScale As ScaleType
Private AnimationTLS() As Integer
'
'END DECLARATION
'=====================================================
==============================================

Public Sub Simulate_Reliability()

   If N_to_N_Detection(False) = False Then 'Checking Node to Node Detection and Creating
Graph Table
        MsgBox "Please verify the node to node detection first!" & vbNewLine & " Then try
again!", vbInformation
        Exit Sub
```

```
    End If
    If ldGenerator.TotalLoaded < 1 Then
        MsgBox "Reliability simulations cannot be performed on no generator", vbInformation
        Exit Sub
    End If
    Dim sm As New Simulation
    sm.Initialize DB
    Dim I As Long, J  As Integer, K As Integer
    For I = 0 To ldLineTransmission.Total
        If ldLineTransmission.IsExist(I + 1) Then
            If LineTranStart(I).BorderColor = vbRed Then sm.Power_TransmissionLine
LALLineTransmission(I)
        End If
    Next
    frmWait.lalTotal = 2 ^ ldGenerator.TotalLoaded + 1
    frmWait.lalCounter = 1
    frmWait.Show
    frmWait.Refresh
    With frmShowSimulationRelaibityResult.grdGenerator
        .Rows = ldGenerator.TotalLoaded + 1
        .Cols = 4
        .Row = 0
        .col = 0
        .Text = "Generator ID"
        .ColWidth(0) = 1200
        .col = 1
        .Text = "AGC(Real Power)"
        .ColWidth(1) = 1500
        .col = 2
        .Text = "Availability"
        .ColWidth(2) = 1200
        .col = 3
        .Text = "Unavailability"
        .ColWidth(3) = 1200

        J = 1
        For I = 0 To ldGenerator.Total '- 1
            If ldGenerator.IsExist(I + 1) Then
                .Row = J
                .col = 0
                .Text = LALGenerator(I)
                .CellAlignment = 1
                .col = 1
                .Text = sm.Power_Generator(LALGenerator(I))
                .CellAlignment = 1
                .col = 2
                .Text = sm.GeneratorAvalibility(LALGenerator(I))
                .CellAlignment = 1
                .col = 3
                .Text = sm.GeneratorUnavalibility(LALGenerator(I))
                .CellAlignment = 1
                J = J + 1
            End If
        Next
    End With
    Dim BinomialCombinations As New DecToBin
    With frmShowSimulationRelaibityResult.grdState
        .Rows = 2 ^ ldGenerator.TotalLoaded + 1
        .Cols = ldGenerator.TotalLoaded + 5
```

```
.Row = 0
.col = 0
.Text = "No."
.ColWidth(0) = 600
.col = 1
.Text = "State (Probability)"
.ColWidth(1) = 1700
.col = 2
.Text = "AGC"
.ColWidth(2) = 1200
.col = 3
.Text = "E" ' (Energy Loss)"
.ColWidth(3) = 1000
.col = 4
.Text = "ENS"
.ColWidth(4) = 1200
J = 5
For I = 1 To ldGenerator.Total + 1
   If ldGenerator.IsExist(CInt(I)) Then
      .col = J
      .ColWidth(J) = 450
      .Text = LALGenerator(I - 1)
      J = J + 1
   End If
Next

Dim Prob As Double, AGC As Double, TotalLoadFlow As Double
TotalLoadFlow = 0
For I = 1 To ldFeeder.Total + 1
   If ldFeeder.IsExist(CInt(I)) Then
      TotalLoadFlow = TotalLoadFlow + sm.Power_Feeder(LALFeeder(I - 1))
   End If
Next

For I = 0 To 2 ^ ldGenerator.TotalLoaded - 1
   K = 4
   Prob = 1
   AGC = 0
   .Row = I + 1
   .col = 0
   .Text = I + 1
   .CellAlignment = 1
   BinomialCombinations.ChangeToBinary CLng(I)
   For J = 1 To ldGenerator.Total + 1
      If ldGenerator.IsExist(J) Then
         K = K + 1
         .col = K
         If BinomialCombinations.Bit(ldGenerator.TotalLoaded - K + 5) = False Then
            .Text = "A"
            .CellAlignment = 3
            Prob = Prob * sm.GeneratorAvalibility(LALGenerator(J - 1))
            AGC = AGC + sm.Power_Generator(LALGenerator(J - 1))
         Else
            .Text = "U"
            .CellAlignment = 3
            Prob = Prob * sm.GeneratorUnavalibility(LALGenerator(J - 1))
         End If
      End If
   Next
```

```
            .col = 1
            .Text = Format(Prob, "0.000000000000")
            '.CellAlignment = 1
            .col = 2
            .Text = Format(AGC, "0.0#######")
            '.CellAlignment = 1
            .col = 3
            If AGC > TotalLoadFlow Then
               .Text = 0
               .col = 4
               .Text = 0
               '.CellAlignment = 1
               .col = 3
            Else
               .Text = TotalLoadFlow - AGC
               .col = 4
               .Text = Val(TotalLoadFlow - AGC) * 8760
               '.CellAlignment = 1
               .col = 3
            End If
            '.CellAlignment = 1
            frmWait.lalCounter = frmWait.lalCounter + 1
            frmWait.lalCounter.Refresh
         Next
      End With
      Dim EENS As Double, temp1 As Double, temp2 As Double
      EENS = 0
      With frmShowSimulationRelaibityResult
         For I = 1 To .grdState.Rows - 1
            .grdState.Row = I
            .grdState.col = 1
            temp1 = .grdState.Text
            .grdState.col = 4
            temp2 = .grdState.Text
            If temp2 <> 0 Then EENS = EENS + temp1 * temp2
         Next
         .lalTotalFlow = "Load Total  = " & TotalLoadFlow
         .lalExepectedLossE.Caption = "EENS = " & Format(EENS, "#0.0##")
         .ResetControls
         .Caption = "Relaibity Simulations Result  of " & Caption
         .Show
      End With
      'Dim MatrixReliability As New Matrix
      'MatrixA.Order ldLineBusbar.TotalLoaded - 1, ldLineBusbar.TotalLoaded - 1
      'MatrixB.Order ldLineBusbar.TotalLoaded - 1, 1
      'frmWait.Show

End Sub
Public Sub DisplayFlow()
   On Error Resume Next
   'This function initialize the flow direction
   If N_to_N_Detection(False) = False Then 'Checking Node to Node Detection and Creating
Graph Table
      MsgBox "Please verify the node to node detection first!" & vbNewLine & " Then try
again!", vbInformation
      Exit Sub
   End If
   Dim matrixa As New Matrix
   Dim matrixb As New Matrix
```

```
    Dim TempIndexes() As Integer, K As Integer
    Dim I As Integer, J As Integer, rowno As Integer, colno As Integer, M As Integer
    Dim adja As New Matrix, mode As Double
    Dim thitai As Double, thitaj As Double
    Dim busbari As String, busbarj As String

    Me.Cls

    Dim sm As New Simulation
    sm.Initialize DB
    If ldLineBusbar.TotalLoaded < 2 Then
        ReDim AnimationTLS(ldLineTransmission.Total)
        For I = 0 To ldLineTransmission.Total   'Checking all lines
            If ldLineTransmission.IsExist(I + 1) Then   'If the transmission line exist
                For K = 1 To frmGraphTable.grdGraph.Rows - 1
                    If frmGraphTable.grdGraph.TextMatrix(K, 1) = LALLineTransmission(I) Then
                        busbari = frmGraphTable.grdGraph.TextMatrix(K, 2)
                        busbarj = frmGraphTable.grdGraph.TextMatrix(K, 3)
                        Exit For
                    End If
                Next
                If InStr(busbari, "B") = 1 Then
                    AnimationTLS(I) = -1
                ElseIf InStr(busbarj, "B") = 1 Then
                    AnimationTLS(I) = 1
                Else    'If both ends of transmission line is not connected to any busbar ??
                    AnimationTLS(I) = sm.Tran_Line_Direction(LALLineTransmission(I))
                End If
            End If
        Next
        TimerSimulationFlow.Enabled = True
        Exit Sub
    End If
    matrixa.Order ldLineBusbar.TotalLoaded - 1, ldLineBusbar.TotalLoaded - 1
    matrixb.Order ldLineBusbar.TotalLoaded - 1, 1
    frmWait.Show
    frmWait.lalTotal = matrixb.TotalRows * matrixb.TotalRows + 5
    frmWait.lalCounter = 0
    frmWait.Refresh


    For I = 0 To ldLineTransmission.Total   'IF the transmission line is failed
        If ldLineTransmission.IsExist(I + 1) Then
            If LineTranStart(I).BorderColor = vbRed Then sm.Power_TransmissionLine
LALLineTransmission(I)
        End If
    Next

    'Inserting values to Matices for simulations
    '*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
    rowno = 0
    For I = 0 To ldLineBusbar.Total - 1
        If ldLineBusbar.IsExist(I + 1) Then
        rowno = rowno + 1
            colno = 0
            If rowno > matrixa.TotalRows Then Exit For
            For J = 0 To ldLineBusbar.Total - 1
                If ldLineBusbar.IsExist(J + 1) Then
                    colno = colno + 1
```

```
                If colno > matrixa.TotalCols Then Exit For
                If rowno = colno Then
                    Dim K3 As Integer
                    For K3 = 0 To ldLineBusbar.Total
                        If ldLineBusbar.IsExist(K3 + 1) Then
                            matrixa.ChangeValue(rowno, colno) = matrixa.GetValue(rowno, colno) + sm.Tran_Line_Power(LALLineBusbar(I).Caption, LALLineBusbar(K3).Caption)
                        End If
                    Next
                Else
                    matrixa.ChangeValue(rowno, colno) = -1 * sm.Tran_Line_Power(LALLineBusbar(I).Caption, LALLineBusbar(J).Caption)
                End If
            End If
        Next
        matrixb.ChangeValue(rowno, 1) = sm.BusBar_Net_Power(LALLineBusbar(I).Caption)
    End If
Next
'*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/


frmWait.lalCounter = frmWait.lalCounter + 1
frmWait.lalCounter.Refresh
mode = matrixa.Determinent
frmWait.lalCounter = frmWait.lalCounter + 1
frmWait.lalCounter.Refresh
If mode = 0 Then
    frmWait.lalCounter = frmWait.lalTotal
    MsgBox "Unique solution of the system is not possible!" & vbNewLine & "So flow cannot be identified"
    Exit Sub
End If
Set adja = matrixa.AdjointMatrix
Dim D As New Matrix
frmWait.lalCounter = frmWait.lalCounter + 1
frmWait.lalCounter.Refresh
Set D = MatMultiply(adja, matrixb)
frmWait.lalCounter = frmWait.lalCounter + 1

For I = 1 To D.TotalRows
    For J = 1 To D.TotalCols
        D.ChangeValue(I, J) = D.GetValue(I, J) / mode
    Next
Next
frmWait.lalCounter = frmWait.lalCounter + 1
frmWait.lalCounter.Refresh

ReDim AnimationTLS(ldLineTransmission.Total)
ReDim TempIndexes(ldLineBusbar.Total)
J = 1
For I = 0 To ldLineBusbar.Total
    TempIndexes(I) = -1
    If ldLineBusbar.IsExist(I + 1) Then
        TempIndexes(I) = J
        K = I   'Keep the last index
        J = J + 1
    End If
Next
TempIndexes(K) = -1
```

```
M = 0
J = 0   'It is used to hold the current solution matrix row
For I = 0 To ldLineTransmission.Total   'Checking all lines
   If ldLineTransmission.IsExist(I + 1) Then   'If the transmission line exist
      For K = 1 To frmGraphTable.grdGraph.Rows - 1
         If frmGraphTable.grdGraph.TextMatrix(K, 1) = LALLineTransmission(I) Then
            busbari = frmGraphTable.grdGraph.TextMatrix(K, 2)
            busbarj = frmGraphTable.grdGraph.TextMatrix(K, 3)
            Exit For
         End If
      Next
      If InStr(busbarj, "B") = 1 And InStr(busbari, "B") = 1 Then
         busbari = Right(frmGraphTable.grdGraph.TextMatrix(K, 2),
Len(frmGraphTable.grdGraph.TextMatrix(K, 2)) - 1)
         busbarj = Right(frmGraphTable.grdGraph.TextMatrix(K, 3),
Len(frmGraphTable.grdGraph.TextMatrix(K, 3)) - 1)
         thitai = 0
         thitaj = 0
         If TempIndexes(busbari - 1) > -1 Then thitai =
D.GetValue(TempIndexes(busbari - 1), 1)
         If TempIndexes(busbarj - 1) > -1 Then thitaj =
D.GetValue(TempIndexes(busbarj - 1), 1)
         If TempIndexes(busbarj - 1) = -1 Then thitaj = 0
         If thitai - thitaj = 0 Or sm.Power_TransmissionLine(LALLineTransmission(I)) = 0
Then
            AnimationTLS(I) = 0
         ElseIf (thitai - thitaj) * sm.Power_TransmissionLine(LALLineTransmission(I)) > 0
Then
            If (thitai - thitaj) * sm.Power_TransmissionLine(LALLineTransmission(I)) <
sm.Cap_TransmissionLine(LALLineTransmission(I)) Then
               AnimationTLS(I) = 1
            Else
               AnimationTLS(I) = 2
            End If
         ElseIf (thitai - thitaj) * sm.Power_TransmissionLine(LALLineTransmission(I)) < 0
Then
            If Abs((thitai - thitaj) * sm.Power_TransmissionLine(LALLineTransmission(I)))
> sm.Cap_TransmissionLine(LALLineTransmission(I)) Then
               AnimationTLS(I) = -2
            Else
               AnimationTLS(I) = -1
            End If
         End If
      Load LALPowerFlow(I)
      Load LALlineloss(I)
      LALPowerFlow(I).Visible = False
      LALPowerFlow(I) = "PowerFlow = " & Round((thitai - thitaj) *
sm.Power_TransmissionLine(LALLineTransmission(I)), 2)
      LALPowerFlow(I).Move LALLineTransmission(I).Left +
LALLineTransmission(I).Width * 0.5 - LALPowerFlow(I).Width * 0.5,
LALLineTransmission(I).Top + LALPowerFlow(I).Height - 40
      LALPowerFlow(I).Visible = True
      LALlineloss(I) = "Line Loss"
      LALlineloss(I).Visible = False
      ElseIf InStr(busbari, "B") = 1 Then
         AnimationTLS(I) = -1
      ElseIf InStr(busbarj, "B") = 1 Then
         AnimationTLS(I) = 1
```

```
            Else    'If both ends of transmission line is not connected to any busbar ??
                AnimationTLS(I) = sm.Tran_Line_Direction(LALLineTransmission(I))
            End If
        Else
            LALPowerFlow(I).Visible = False
            LALPowerFlow(ldLineTransmission.TotalLoaded).Visible = False
        End If
    Next
"'**Change here
For M = 0 To ldGenerator.Total
    If ldGenerator.IsExist(M + 1) Then
            Load LALGen(J)
            LALGen(J).Visible = False
            'LALGen(J) = sm.Power_Generator(LALGenerator(M)) & "PU"
            LALGen(J) = sm.Power_Generator(LALGenerator(M))
            LALGen(J).Move LALGenerator(M).Left + LALGenerator(M).Width * 0.5 -
LALGen(J).Width * 0.5, LALGenerator(M).Top + LALGen(J).Height - 40
            LALGen(J).Visible = True
    Else
            LALGen(J).Visible = False
            LALGen(ldGenerator.TotalLoaded).Visible = False
    End If
            J = J + 1
Next
J = 0
For M = 0 To ldFeeder.Total
    If ldFeeder.IsExist(M + 1) Then
            Load Lalfee(J)
            Lalfee(J).Visible = False
            'Lalfee(J) = sm.Power_Feeder(LALFeeder(M)) & "PU"
            Lalfee(J) = sm.Power_Feeder(LALFeeder(M))
            Lalfee(J).Move LALFeeder(M).Left + LALFeeder(M).Width * 0.5 - Lalfee(J).Width *
0.5, LALFeeder(M).Top + Lalfee(J).Height - 40
            Lalfee(J).Visible = True
    Else
            Lalfee(J).Visible = False
            Lalfee(ldFeeder.TotalLoaded).Visible = False
    End If
            J = J + 1
Next
J = 0
For M = 0 To ldtransformer.Total
    If ldtransformer.IsExist(M + 1) Then
            Load LalInfo(J)
            LalInfo(J).Visible = False
            'LalInfo(J) = sm.Power_Transformer(LALTransformer(M)) & "PU"
            LalInfo(J) = sm.Power_Transformer(LALTransformer(M))
            LalInfo(J).Move LALTransformer(M).Left + LALTransformer(M).Width * 0.5 -
LalInfo(J).Width * 0.5, LALTransformer(M).Top + LalInfo(J).Height - 40
            LalInfo(J).Visible = True
    Else
            LalInfo(J).Visible = False
            LalInfo(ldtransformer.TotalLoaded).Visible = False
    End If
            J = J + 1
Next
J = 0
For M = 0 To ldLineBusbar.Total
    If ldLineBusbar.IsExist(M + 1) Then
```

```
        Load Lallinebus(J)
        Lallinebus(J) = "linebus"
        Lallinebus(J).Visible = False
        J = J + 1
    End If
Next
'''***
    TimerSimulationFlow.Enabled = True
End Sub
Public Sub Simulate_PerformanceIndicesCal()

    If N_to_N_Detection(False) = False Then 'Checking Node to Node Detection and Creating
Graph Table
        MsgBox "Please verify the node to node detection first!" & vbNewLine & " Then try
again!", vbInformation
        Exit Sub
    End If
    If ldGenerator.TotalLoaded < 1 Then
        MsgBox "Reliability simulations cannot be performed on no generator", vbInformation
        Exit Sub
    End If
    Dim sm As New Simulation
    sm.Initialize DB
    Dim I As Long, J  As Integer, K As Integer
    For I = 0 To ldLineTransmission.Total
        If ldLineTransmission.IsExist(I + 1) Then
            If  LineTranStart(I).BorderColor  =  vbRed  Then  sm.ZeroPower_TransmissionLine
LALLineTransmission(I)
        End If
    Next
    'The difference of the Code of Relaibility simulation and it is in "="=
    '''="="="="="="="="="="="="="="="="="="="="="="="="="="="="="="="=
    Dim TRTime As Double, TFRecords As Double, TGs As Long, TFT As Double, TotalAGC As
Double
    Dim Avalibility As Double, UnAvalibility As Double
    If DisplayPriod = True Then
        Dim StartDate As New LocalDate, EndDate As New LocalDate
        On Error Resume Next
        StartDate.Value = FrmPerformanceIndecCal.cboStartDate
        EndDate.Value = FrmPerformanceIndecCal.CboEndDate
        Set rsForAll = DB.OpenRecordset("Generators Failures")
        rsForAll.MoveLast
        Dim TempDate As New LocalDate
        TFRecords = rsForAll.RecordCount
        TRTime = 0
        rsForAll.MoveFirst
        For I = 0 To rsForAll.RecordCount - 1
            TempDate.Value = rsForAll.Fields("FailureDate")
            If          StartDate.CompareLocalDate(TempDate)          <=          0          And
EndDate.CompareLocalDate(TempDate) >= 0 Then
                TRTime = TRTime + Val(rsForAll.Fields("RepairTime"))
            End If
            rsForAll.MoveNext
        Next
        Set rsForAll = DB.OpenRecordset("Generators")
        rsForAll.MoveLast
        TGs = rsForAll.RecordCount
        TotalAGC = 0
        rsForAll.MoveFirst
```

```
    For I = 0 To TGs - 1
        TotalAGC = TotalAGC + Val(rsForAll.Fields("RealPower"))
        rsForAll.MoveNext
    Next
Else
    Exit Sub
End If
With frmShowSimulationRelaibityResult.grdGenerator
    .Rows = 2
    .Cols = 4
    .Row = 0
    .col = 0
    .Text = "Component"
    .ColWidth(0) = 1200
    .col = 1
    .Text = "Total AGC"
    .ColWidth(1) = 1500
    .col = 2
    .Text = "Availability"
    .ColWidth(2) = 1200
    .col = 3
    .Text = "Unavailability"
    .ColWidth(3) = 1200
    .Row = 1
    .col = 0
    .Text = "System"
    .CellAlignment = 1
    .col = 1
    .Text = TotalAGC
    .CellAlignment = 1
    .col = 2
    Dim FailureRate As Double, RepairRate As Double
    FailureRate = TFRecords / TGs
    RepairRate = TRTime / (TFRecords * TGs)
    UnAvalibility = RepairRate / CDbl(FailureRate + RepairRate)
    .Text = Format(UnAvalibility, "0.00000000")
    .CellAlignment = 1
    .col = 3
    Avalibility = FailureRate / CDbl(FailureRate + RepairRate)
    .Text = Format(Avalibility, "0.00000000")
    .CellAlignment = 1
End With
frmShowSimulationRelaibityResult.Label1 = "System Information"
With frmShowSimulationRelaibityResult.grdState
    .Rows = 3
    .Cols = 6
    .Row = 0
    .col = 0
    .Text = "No."
    .ColWidth(0) = 600
    .col = 1
    .Text = "State (Probability)"
    .ColWidth(1) = 1700
    .col = 2
    .Text = "AGC"
    .ColWidth(2) = 1200
    .col = 3
    .Text = "E" ' (Energy Loss)"
    .ColWidth(3) = 1000
```

```
        .col = 4
        .Text = "ENS"
        .ColWidth(4) = 1200
        .col = 5
        .ColWidth(5) = 450

        Dim Prob As Double, AGC As Double, TotalLoadFlow As Double
        TotalLoadFlow = 0
        For I = 1 To ldFeeder.Total + 1
            If ldFeeder.IsExist(CInt(I)) Then
                TotalLoadFlow = TotalLoadFlow + sm.Power_Feeder(LALFeeder(I - 1))
            End If
        Next
        .TextMatrix(1, 0) = 1
        .TextMatrix(2, 0) = 2
        .TextMatrix(1, 1) = Format(Avalibility, "0.00000000")
        .TextMatrix(2, 1) = Format(UnAvalibility, , "0.00000000")

        .TextMatrix(1, 2) = TotalAGC
        .TextMatrix(2, 2) = 0

        .TextMatrix(1, 3) = 0
        If TotalLoadFlow - TotalAGC > 0 Then .TextMatrix(1, 3) = TotalLoadFlow - TotalAGC
        .TextMatrix(2, 3) = TotalLoadFlow
        .TextMatrix(1, 4) = .TextMatrix(1, 3) * 8760
        .TextMatrix(2, 4) = .TextMatrix(2, 3) * 8760
        .TextMatrix(1, 5) = "A"
        .TextMatrix(2, 5) = "U"

    End With
    Dim EENS As Double
    EENS = 0
    With frmShowSimulationRelaibityResult
        EENS = Val(frmShowSimulationRelaibityResult.grdState.TextMatrix(2, 4) *
frmShowSimulationRelaibityResult.grdGenerator.TextMatrix(1, 3))
        .lalTotalFlow = "Load Total  = " & TotalLoadFlow
        .lalExepectedLossE.Caption = "EENS = " & Format(EENS, "#0.0##")
        .ResetControls
        .Caption = "Relaibity Simulations Result  of " & Caption
        .Show
    End With

End Sub
Public Sub Combi(Optional showresult As Boolean = True)
On Error Resume Next

    If N_to_N_Detection(False) = False Then
        MsgBox "Please verify the node to node detection first!" & vbNewLine & " Then try
again!", vbInformation
        Exit Sub
    End If

    Dim sm As New Simulation
    sm.Initialize DB
    Dim busdatamatrix As New Matrix
    Dim I As Integer
    Dim f As Integer

    Me.Cls
```

```
busdatamatrix.Order ldLineBusbar.TotalLoaded, 11

For I = 0 To ldLineBusbar.Total
   If ldLineBusbar.IsExist(I + 1) Then
      busdatamatrix.ChangeValue(I + 1, 1) = I + 1
      busdatamatrix.ChangeValue(I + 1, 2) = sm.buscode_busbar(LALLineBusbar(I))
      busdatamatrix.ChangeValue(I + 1, 3) = sm.Voltage_Busbar(LALLineBusbar(I))
      busdatamatrix.ChangeValue(I + 1, 4) = 0

      'checking for feeder
      Set rsForAll = DB.OpenRecordset("Feeder Parameters", 2, 0)
      For f = 0 To ldFeeder.Total
         rsForAll.FindFirst "FeederID='" & LALFeeder(f) & "'"
         If rsForAll.NoMatch = True Then
            MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
            End
         End If
         If rsForAll.Fields("Connected from") = "B" & I + 1 Then
            busdatamatrix.ChangeValue(I + 1, 5) = sm.Power_Feeder(LALFeeder(f))
            busdatamatrix.ChangeValue(I + 1, 6) = sm.Reactive_Feeder(LALFeeder(f))
         End If
      Next

      'checking for generator
      Set rsForAll = DB.OpenRecordset("Generators", 2, 0)
      For f = 0 To ldGenerator.Total
         rsForAll.FindFirst "GeneratorsID='" & LALGenerator(f) & "'"
         If rsForAll.NoMatch = True Then
            MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
            End
         End If
         If rsForAll.Fields("ConnectedtoBusbar") = "B" & I + 1 Then
            busdatamatrix.ChangeValue(I + 1, 7) =
sm.Power_Generator(LALGenerator(f))
            busdatamatrix.ChangeValue(I + 1, 8) =
sm.Reactive_Generator(LALGenerator(f))
            busdatamatrix.ChangeValue(I + 1, 9) =
sm.Qmin_Generator(LALGenerator(f))
            busdatamatrix.ChangeValue(I + 1, 10) =
sm.Qmax_Generator(LALGenerator(f))
         End If
      Next
      busdatamatrix.ChangeValue(I + 1, 11) = 0
   End If
Next

Dim linedatamatrix As New Matrix
Dim last As Integer
Dim first As Integer
Dim count As Integer
Dim transIndex As String

Set rsForAll = DB.OpenRecordset("Transmission Lines", 2, 0)
ReDim AnimationTLS(ldLineTransmission.Total)
rsForAll.MoveLast
count = rsForAll.RecordCount

rsForAll.MoveFirst
```

```
    linedatamatrix.Order count, 7
    f = 0
    For I = 0 To count - 1

        If ldLineTransmission.IsExist(I + 1) Then
            linedatamatrix.ChangeValue(I + 1, 1) = Right(rsForAll.Fields("Connected From"), 1)
            linedatamatrix.ChangeValue(I + 1, 2) = Right(rsForAll.Fields("Connected To"), 1)
            transIndex = rsForAll.Fields("TransmissionLinesID")
            linedatamatrix.ChangeValue(I + 1, 3) =
sm.Reactance_TransmissionLine(transIndex)
            linedatamatrix.ChangeValue(I + 1, 4) = sm.Imagine_TransmissionLine(transIndex)
            linedatamatrix.ChangeValue(I + 1, 5) = 0
            linedatamatrix.ChangeValue(I + 1, 6) = 1
            linedatamatrix.ChangeValue(I + 1, 7) = Right(transIndex, Len(transIndex) - 2)
        End If
    rsForAll.Move 1
    f = f + 1
    Next

"""" LFYBUS """"
    Dim zrect As New Matrix
    Dim zpolar As New Matrix
    Dim ypolar As New Matrix
    Dim yrect As New Matrix
    Dim ybusreal As New Matrix
    Dim ybusimg As New Matrix
    Dim ybusrealpolar As New Matrix
    Dim ybusangpolar As New Matrix
    Dim J As Integer
    Dim A As Integer

    zpolar.Order count, 2
    zrect.Order count, 2
    ypolar.Order count, 2
    yrect.Order count, 2
    ybusreal.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
    ybusimg.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
    ybusrealpolar.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
    ybusangpolar.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded

    For I = 0 To count - 1
        zrect.ChangeValue(I + 1, 1) = linedatamatrix.GetValue(I + 1, 3)
        zrect.ChangeValue(I + 1, 2) = linedatamatrix.GetValue(I + 1, 4)
        zpolar.ChangeValue(I + 1, 1) = Sqr(linedatamatrix.GetValue(I + 1, 3) ^ 2 +
linedatamatrix.GetValue(I + 1, 4) ^ 2)
        zpolar.ChangeValue(I + 1, 2) = Atn(linedatamatrix.GetValue(I + 1, 4) /
linedatamatrix.GetValue(I + 1, 3)) * 180 / 3.14159265358979
        ypolar.ChangeValue(I + 1, 1) = 1 / zpolar.GetValue(I + 1, 1)
        ypolar.ChangeValue(I + 1, 2) = 0 - zpolar.GetValue(I + 1, 2)
        yrect.ChangeValue(I + 1, 1) = ypolar.GetValue(I + 1, 1) * Cos(ypolar.GetValue(I + 1,
2) * 3.14159265358979 / 180)
        yrect.ChangeValue(I + 1, 2) = ypolar.GetValue(I + 1, 1) * Sin(ypolar.GetValue(I + 1,
2) * 3.14159265358979 / 180)
    Next

    For I = 0 To ldLineBusbar.TotalLoaded - 1
        For J = 0 To ldLineBusbar.TotalLoaded - 1
            ybusreal.ChangeValue(I + 1, J + 1) = 0
            ybusimg.ChangeValue(I + 1, J + 1) = 0
```

```
      Next
   Next

   For I = 0 To count - 1
      ybusreal.ChangeValue(linedatamatrix.GetValue(I + 1, 1), linedatamatrix.GetValue(I +
1, 2)) = 0 - yrect.GetValue(I + 1, 1)
      ybusimg.ChangeValue(linedatamatrix.GetValue(I + 1, 1), linedatamatrix.GetValue(I +
1, 2)) = 0 - yrect.GetValue(I + 1, 2)
      ybusreal.ChangeValue(linedatamatrix.GetValue(I + 1, 2), linedatamatrix.GetValue(I +
1, 1)) = ybusreal.GetValue(linedatamatrix.GetValue(I + 1, 1), linedatamatrix.GetValue(I +
1, 2))
      ybusimg.ChangeValue(linedatamatrix.GetValue(I + 1, 2), linedatamatrix.GetValue(I +
1, 1)) = ybusimg.GetValue(linedatamatrix.GetValue(I + 1, 1), linedatamatrix.GetValue(I +
1, 2))
   Next

   For I = 1 To ldLineBusbar.TotalLoaded
      For J = 1 To count
         If linedatamatrix.GetValue(J, 1) = I Then
            ybusreal.ChangeValue(I, I) = ybusreal.GetValue(I, I) + yrect.GetValue(J, 1)
            ybusimg.ChangeValue(I, I) = ybusimg.GetValue(I, I) + yrect.GetValue(J, 2)
         ElseIf linedatamatrix.GetValue(J, 2) = I Then
               ybusreal.ChangeValue(I, I) = ybusreal.GetValue(I, I) + yrect.GetValue(J, 1)
               ybusimg.ChangeValue(I, I) = ybusimg.GetValue(I, I) + yrect.GetValue(J, 2)
         End If
      Next
      For A = 1 To ldLineBusbar.TotalLoaded
         ybusrealpolar.ChangeValue(I, A) = Sqr(ybusreal.GetValue(I, A) ^ 2 +
ybusimg.GetValue(I, A) ^ 2)
         If ybusimg.GetValue(I, A) < 0 And ybusreal.GetValue(I, A) < 0 Then
            ybusangpolar.ChangeValue(I, A) = Atn(ybusimg.GetValue(I, A) /
ybusreal.GetValue(I, A)) * 180 / 3.14159265358979 - 180
         ElseIf ybusimg.GetValue(I, A) > 0 And ybusreal.GetValue(I, A) < 0 Then
               ybusangpolar.ChangeValue(I, A) = Atn(ybusimg.GetValue(I, A) /
ybusreal.GetValue(I, A)) * 180 / 3.14159265358979 + 180
            Else
               ybusangpolar.ChangeValue(I, A) = Atn(ybusimg.GetValue(I, A) /
ybusreal.GetValue(I, A)) * 180 / 3.14159265358979
         End If
      Next
   Next

""" LFGAUSS """
Dim vreal As New Matrix
Dim vimg As New Matrix
Dim vrealpolar As New Matrix
Dim vangpolar As New Matrix
Dim vm As New Matrix
Dim delta As New Matrix
Dim deltad As New Matrix
Dim pg As New Matrix
Dim pd As New Matrix
Dim p As New Matrix
Dim qd As New Matrix
Dim qg As New Matrix
Dim q As New Matrix
Dim N As Integer
Dim kb As New Matrix
Dim dv As New Matrix
```

```
vm.Order ldLineBusbar.TotalLoaded, 1
delta.Order ldLineBusbar.TotalLoaded, 1
deltad.Order ldLineBusbar.TotalLoaded, 1
vreal.Order ldLineBusbar.TotalLoaded, 1
vimg.Order ldLineBusbar.TotalLoaded, 1
pg.Order ldLineBusbar.TotalLoaded, 1
vrealpolar.Order ldLineBusbar.TotalLoaded, 1
vangpolar.Order ldLineBusbar.TotalLoaded, 1
pd.Order ldLineBusbar.TotalLoaded, 1
p.Order ldLineBusbar.TotalLoaded, 1
qd.Order ldLineBusbar.TotalLoaded, 1
qg.Order ldLineBusbar.TotalLoaded, 1
q.Order ldLineBusbar.TotalLoaded, 1
kb.Order ldLineBusbar.TotalLoaded, 1
dv.Order ldLineBusbar.TotalLoaded, 1

   For I = 1 To ldLineBusbar.TotalLoaded
      N = busdatamatrix.GetValue(I, 1)
      vm.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 3)
      delta.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 4)
      kb.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 2)
      pg.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 7)
      pd.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 5)
      qd.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 6)
      qg.ChangeValue(N, 1) = busdatamatrix.GetValue(I, 8)
      If vm.GetValue(N, 1) <= 0 Then
         vm.ChangeValue(N, 1) = 1
         vreal.ChangeValue(N, 1) = 1
         vimg.ChangeValue(N, 1) = 0
         vrealpolar.ChangeValue(N, 1) = 1
         vangpolar.ChangeValue(N, 1) = 1
      Else
         delta.ChangeValue(N, 1) = 3.14159265358979 / 180 * delta.GetValue(N, 1)
         vreal.ChangeValue(N, 1) = vm.GetValue(N, 1) * Cos(delta.GetValue(N, 1) *
3.14159265358979 / 180)
         vimg.ChangeValue(N, 1) = vm.GetValue(N, 1) * Sin(delta.GetValue(N, 1) *
3.14159265358979 / 180)
         vrealpolar.ChangeValue(N, 1) = Sqr(vreal.GetValue(N, 1) ^ 2 + vimg.GetValue(N,
1) ^ 2)
         vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) / vreal.GetValue(N, 1)) *
180 / 3.14159265358979
         p.ChangeValue(N, 1) = (pg.GetValue(N, 1) - pd.GetValue(N, 1)) / 100
         q.ChangeValue(N, 1) = (qg.GetValue(N, 1) - qd.GetValue(N, 1)) / 100
      End If
   dv.GetValue(N, 1) = 0
   Next

Dim vcpolar As New Matrix
Dim vcrect As New Matrix
Dim scpolar As New Matrix
Dim screct As New Matrix
Dim maxerror As Double
Dim accuracy As Double
Dim iter As Integer
Dim maxiter As Integer
Dim yvreal As Double
Dim yvimg As Double
Dim K As Integer
```

```
Dim extrarealpolar As New Matrix
Dim extraimgpolar As New Matrix
Dim extrarealrect As New Matrix
Dim extraimgrect As New Matrix
Dim l As Integer
Dim dp As New Matrix
Dim dq As New Matrix
Dim yload As New Matrix
Dim tableq As Double
Dim tablep As Double
Dim o As Integer
Dim qgc As Double
Dim vci As Double
Dim vcr As Double

vcpolar.Order ldLineBusbar.TotalLoaded, 2
vcrect.Order ldLineBusbar.TotalLoaded, 2
scpolar.Order ldLineBusbar.TotalLoaded, 2
screct.Order ldLineBusbar.TotalLoaded, 2
dq.Order ldLineBusbar.TotalLoaded, 1
dp.Order ldLineBusbar.TotalLoaded, 1
extrarealpolar.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
extraimgpolar.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
extrarealrect.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
extraimgrect.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
yload.Order ldLineBusbar.TotalLoaded, 2

accuracy = 0.001
maxiter = 100
maxerror = 10
iter = 0

   For I = 1 To ldLineBusbar.TotalLoaded
      For J = 1 To ldLineBusbar.TotalLoaded
         vcpolar.ChangeValue(I, 1) = 0
         vcrect.ChangeValue(I, 1) = 0
         vcpolar.ChangeValue(I, 2) = 0
         vcrect.ChangeValue(I, 2) = 0
         scpolar.ChangeValue(I, 1) = 0
         scpolar.ChangeValue(I, 2) = 0
         screct.ChangeValue(I, 1) = 0
         screct.ChangeValue(I, 2) = 0
      Next
   Next

   While maxerror >= accuracy And iter <= maxiter
      iter = iter + 1
      For N = 1 To ldLineBusbar.TotalLoaded
         yvreal = 0
         yvimg = 0
         For l = 1 To count
            If linedatamatrix.GetValue(l, 1) = N Then
               K = linedatamatrix.GetValue(l, 2)
               extrarealpolar.ChangeValue(N, K) = ybusrealpolar.GetValue(N, K) *
vrealpolar.GetValue(K, 1)
               extraimgpolar.ChangeValue(N, K) = ybusangpolar.GetValue(N, K) +
vangpolar.GetValue(K, 1)
               extrarealrect.ChangeValue(N, K) = extrarealpolar.GetValue(N, K) *
Cos(extraimgpolar.GetValue(N, K) * 3.14159265358979 / 180)
```

```
                extraimgrect.ChangeValue(N, K) = extrarealpolar.GetValue(N, K) *
Sin(extraimgpolar.GetValue(N, K) * 3.14159265358979 / 180)
                yvreal = yvreal + extrarealrect.GetValue(N, K)
                yvimg = yvimg + extraimgrect.GetValue(N, K)
            ElseIf linedatamatrix.GetValue(l, 2) = N Then
                K = linedatamatrix.GetValue(l, 1)
                extrarealpolar.ChangeValue(N, K) = ybusrealpolar.GetValue(N, K) *
vrealpolar.GetValue(K, 1)
                extraimgpolar.ChangeValue(N, K) = ybusangpolar.GetValue(N, K) +
vangpolar.GetValue(K, 1)
                extrarealrect.ChangeValue(N, K) = extrarealpolar.GetValue(N, K) *
Cos(extraimgpolar.GetValue(N, K) * 3.14159265358979 / 180)
                extraimgrect.ChangeValue(N, K) = extrarealpolar.GetValue(N, K) *
Sin(extraimgpolar.GetValue(N, K) * 3.14159265358979 / 180)
                yvreal = yvreal + extrarealrect.GetValue(N, K)
                yvimg = yvimg + extraimgrect.GetValue(N, K)
            End If
        Next

        extrarealpolar.ChangeValue(N, N) = ybusrealpolar.GetValue(N, N) *
vrealpolar.GetValue(N, 1)
        extraimgpolar.ChangeValue(N, N) = ybusangpolar.GetValue(N, N) +
vangpolar.GetValue(N, 1)
        extrarealrect.ChangeValue(N, N) = extrarealpolar.GetValue(N, N) *
Cos(extraimgpolar.GetValue(N, N) * 3.14159265358979 / 180)
        extraimgrect.ChangeValue(N, N) = extrarealpolar.GetValue(N, N) *
Sin(extraimgpolar.GetValue(N, N) * 3.14159265358979 / 180)
        scpolar.ChangeValue(N, 1) = vrealpolar.GetValue(N, 1) *
Sqr((extrarealrect.GetValue(N, N) + yvreal) ^ 2 + (extraimgrect.GetValue(N, N) + yvimg) ^
2)
        If (extrarealrect.GetValue(N, N) + yvreal) < 0 And (extraimgrect.GetValue(N, N) +
yvimg) > 0 Then
            scpolar.ChangeValue(N, 2) = -(vangpolar.GetValue(N, 1)) +
(Atn((extraimgrect.GetValue(N, N) + yvimg) / (extrarealrect.GetValue(N, N) + yvreal)) *
180 / 3.14159265358979) + 180
        ElseIf (extrarealrect.GetValue(N, N) + yvreal) < 0 And (extraimgrect.GetValue(N,
N) + yvimg) < 0 Then
            scpolar.ChangeValue(N, 2) = -(vangpolar.GetValue(N, 1)) +
(Atn((extraimgrect.GetValue(N, N) + yvimg) / (extrarealrect.GetValue(N, N) + yvreal)) *
180 / 3.14159265358979) - 180
            Else
                scpolar.ChangeValue(N, 2) = -(vangpolar.GetValue(N, 1)) +
(Atn((extraimgrect.GetValue(N, N) + yvimg) / (extrarealrect.GetValue(N, N) + yvreal)) *
180 / 3.14159265358979)
        End If
        screct.ChangeValue(N, 1) = scpolar.GetValue(N, 1) * Cos(scpolar.GetValue(N, 2) *
3.14159265358979 / 180)
        screct.ChangeValue(N, 2) = -(scpolar.GetValue(N, 1) * Sin(scpolar.GetValue(N, 2) *
3.14159265358979 / 180))

        dp.ChangeValue(N, 1) = p.GetValue(N, 1) - screct.GetValue(N, 1)
        dq.ChangeValue(N, 1) = q.GetValue(N, 1) - screct.GetValue(N, 2)

        If kb.GetValue(N, 1) = 1 Then
            p.ChangeValue(N, 1) = screct.GetValue(N, 1)
            q.ChangeValue(N, 1) = screct.GetValue(N, 2)
            dp.ChangeValue(N, 1) = 0
            dq.ChangeValue(N, 1) = 0
            vcrect.ChangeValue(N, 1) = vreal.GetValue(N, 1)
```

```
            vcrect.ChangeValue(N, 2) = vimg.GetValue(N, 1)
        ElseIf kb.GetValue(N, 1) = 2 Then
            q.ChangeValue(N, 1) = screct.GetValue(N, 2)
            If busdatamatrix.GetValue(N, 10) <> 0 Then
                qgc = q.GetValue(N, 1) * 100 + qd.GetValue(N, 1) -
busdatamatrix.GetValue(N, 11)
                If Abs(dq.GetValue(N, 1)) < 0.005 And iter >= 10 Then
                    If dv.GetValue(N, 1) <= 0.045 Then
                        If qgc < busdatamatrix.GetValue(N, 9) Then
                            vm.ChangeValue(N, 1) = vm.GetValue(N, 1) + 0.005
                            dv.ChangeValue(N, 1) = dv.GetValue(N, 1) + 0.005
                        ElseIf qgc > busdatamatrix.GetValue(N, 10) Then
                            vm.ChangeValue(N, 1) = vm.GetValue(N, 1) - 0.005
                            dv.ChangeValue(N, 1) = dv.GetValue(N, 1) + 0.005
                        End If
                    End If
                End If
            End If
        End If

        If kb.GetValue(N, 1) <> 1 Then
            extrarealpolar.ChangeValue(N, N) = Sqr(p.GetValue(N, 1) ^ 2 + q.GetValue(N,
1) ^ 2) / vrealpolar.GetValue(N, 1)
            If q.GetValue(N, 1) > 0 And p.GetValue(N, 1) < 0 Then
                extraimgpolar.ChangeValue(N, N) = -(Atn(q.GetValue(N, 1) / p.GetValue(N,
1)) * 180 / 3.14159265358979 + 180) + vangpolar.GetValue(N, 1)
            ElseIf q.GetValue(N, 1) < 0 And p.GetValue(N, 1) < 0 Then
                extraimgpolar.ChangeValue(N, N) = -(Atn(q.GetValue(N, 1) /
p.GetValue(N, 1)) * 180 / 3.14159265358979 - 180) + vangpolar.GetValue(N, 1)
            Else
                extraimgpolar.ChangeValue(N, N) = -(Atn(q.GetValue(N, 1) /
p.GetValue(N, 1)) * 180 / 3.14159265358979 + vangpolar.GetValue(N, 1))
            End If
            extrarealrect.ChangeValue(N, N) = extrarealpolar.GetValue(N, N) *
Cos(extraimgpolar.GetValue(N, N) * 3.14159265358979 / 180) - yvreal
            extraimgrect.ChangeValue(N, N) = extrarealpolar.GetValue(N, N) *
Sin(extraimgpolar.GetValue(N, N) * 3.14159265358979 / 180) - yvimg
            extrarealpolar.ChangeValue(N, N) = Sqr(extrarealrect.GetValue(N, N) ^ 2 +
extraimgrect.GetValue(N, N) ^ 2)
            If extrarealrect.GetValue(N, N) < 0 And extraimgrect.GetValue(N, N) > 0 Then
                extraimgpolar.ChangeValue(N, N) = (Atn(extraimgrect.GetValue(N, N) /
extrarealrect.GetValue(N, N)) * 180 / 3.14159265358979) + 180
            ElseIf extrarealrect.GetValue(N, N) < 0 And extraimgrect.GetValue(N, N) < 0
Then
                extraimgpolar.ChangeValue(N, N) = (Atn(extraimgrect.GetValue(N, N) /
extrarealrect.GetValue(N, N)) * 180 / 3.14159265358979) - 180
            Else
                extraimgpolar.ChangeValue(N, N) = (Atn(extraimgrect.GetValue(N, N) /
extrarealrect.GetValue(N, N)) * 180 / 3.14159265358979)
            End If
            vcpolar.ChangeValue(N, 1) = extrarealpolar.GetValue(N, N) /
ybusrealpolar.GetValue(N, N)
            vcpolar.ChangeValue(N, 2) = extraimgpolar.GetValue(N, N) -
ybusangpolar.GetValue(N, N)
            vcrect.ChangeValue(N, 1) = vcpolar.GetValue(N, 1) * Cos(vcpolar.GetValue(N,
2) * 3.14159265358979 / 180)
            vcrect.ChangeValue(N, 2) = vcpolar.GetValue(N, 1) * Sin(vcpolar.GetValue(N, 2)
* 3.14159265358979 / 180)
        End If
```

```
        If kb.GetValue(N, 1) = 0 Then
            vreal.ChangeValue(N, 1) = vreal.GetValue(N, 1) + 1.3 * (vcrect.GetValue(N, 1) -
vreal.GetValue(N, 1))
            vimg.ChangeValue(N, 1) = vimg.GetValue(N, 1) + 1.3 * (vcrect.GetValue(N, 2) -
vimg.GetValue(N, 1))
            vrealpolar.ChangeValue(N, 1) = Sqr(vreal.GetValue(N, 1) ^ 2 +
vimg.GetValue(N, 1) ^ 2)
            If vreal.GetValue(N, 1) < 0 And vimg.GetValue(N, 1) > 0 Then
                vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) / vreal.GetValue(N,
1)) * 180 / 3.14159265358979 + 180
            ElseIf vreal.GetValue(N, 1) < 0 And vimg.GetValue(N, 1) < 0 Then
                    vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) /
vreal.GetValue(N, 1)) * 180 / 3.14159265358979 - 180
                Else
                    vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) /
vreal.GetValue(N, 1)) * 180 / 3.14159265358979
                End If
        ElseIf kb.GetValue(N, 1) = 2 Then
                vci = vcrect.GetValue(N, 2)
                vcr = Sqr(vm.GetValue(N, 1) ^ 2 - vci ^ 2)
                vcrect.ChangeValue(N, 1) = vcr
                vcrect.ChangeValue(N, 2) = vci
                vreal.ChangeValue(N, 1) = vreal.GetValue(N, 1) + 1.3 * (vcrect.GetValue(N,
1) - vreal.GetValue(N, 1))
                vimg.ChangeValue(N, 1) = vimg.GetValue(N, 1) + 1.3 * (vcrect.GetValue(N,
2) - vimg.GetValue(N, 1))
                vrealpolar.ChangeValue(N, 1) = Sqr(vreal.GetValue(N, 1) ^ 2 +
vimg.GetValue(N, 1) ^ 2)
                If vreal.GetValue(N, 1) < 0 And vimg.GetValue(N, 1) > 0 Then
                    vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) /
vreal.GetValue(N, 1)) * 180 / 3.14159265358979 + 180
                ElseIf vreal.GetValue(N, 1) < 0 And vimg.GetValue(N, 1) < 0 Then
                        vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) /
vreal.GetValue(N, 1)) * 180 / 3.14159265358979 - 180
                    Else
                        vangpolar.ChangeValue(N, 1) = Atn(vimg.GetValue(N, 1) /
vreal.GetValue(N, 1)) * 180 / 3.14159265358979
                End If
        End If

    Next

o = 1
tablep = Abs(dp.GetValue(o, 1))
For o = 2 To ldLineBusbar.TotalLoaded
    If tablep < Abs(dp.GetValue(o, 1)) Then
        tablep = Abs(dp.GetValue(o, 1))
    End If
Next

o = 1
tableq = Abs(dq.GetValue(o, 1))
For o = 2 To ldLineBusbar.TotalLoaded
    If tableq < Abs(dq.GetValue(o, 1)) Then
        tableq = Abs(dq.GetValue(o, 1))
    End If
Next
```

```
   If tableq > tablep Then
      maxerror = tableq
   Else
      maxerror = tablep
   End If

   Wend

   K = 0
   For N = 1 To ldLineBusbar.TotalLoaded
      vm.ChangeValue(N, 1) = Abs(vrealpolar.GetValue(N, 1))
      deltad.ChangeValue(N, 1) = vangpolar.GetValue(N, 1)
      If kb.GetValue(N, 1) = 1 Then
         pg.ChangeValue(N, 1) = p.GetValue(N, 1) * 100 + pd.GetValue(N, 1)
         qg.ChangeValue(N, 1) = q.GetValue(N, 1) * 100 + qd.GetValue(N, 1) -
busdatamatrix.GetValue(N, 11)
         K = K + 1
      ElseIf kb.GetValue(N, 1) = 2 Then
            K = K + 1
            qg.ChangeValue(N, 1) = q.GetValue(N, 1) * 100 + qd.GetValue(N, 1) -
busdatamatrix.GetValue(N, 11)
      End If
      yload.ChangeValue(N, 1) = pd.GetValue(N, 1) / (100 * vm.GetValue(N, 1) ^ 2)
      yload.ChangeValue(N, 2) = -qd.GetValue(N, 1) / (100 * vm.GetValue(N, 1) ^ 2)
   Next


"" LINEFLOW ""
Dim inrect As New Matrix
Dim inpolar As New Matrix
Dim ikrect As New Matrix
Dim ikpolar As New Matrix
Dim snkpolar As New Matrix
Dim sknpolar As New Matrix
Dim snkrect As New Matrix
Dim sknrect As New Matrix
Dim slrect As New Matrix
Dim sltrect As New Matrix
Dim transnum As Integer

inrect.Order 1, 2
inpolar.Order 1, 2
ikrect.Order 1, 2
ikpolar.Order 1, 2
snkpolar.Order 1, 2
sknpolar.Order 1, 2
snkrect.Order 1, 2
sknrect.Order 1, 2
slrect.Order 1, 2
sltrect.Order 1, 2

   For N = 1 To ldLineBusbar.TotalLoaded
      For l = 1 To count
         If linedatamatrix.GetValue(l, 1) = N Then
            K = linedatamatrix.GetValue(l, 2)
            inrect.ChangeValue(1, 1) = vreal.GetValue(N, 1) - vreal.GetValue(K, 1)
            inrect.ChangeValue(1, 2) = vimg.GetValue(N, 1) - vimg.GetValue(K, 1)
            inpolar.ChangeValue(1, 1) = Sqr(inrect.GetValue(1, 1) ^ 2 + inrect.GetValue(1,
2) ^ 2) * ypolar.GetValue(l, 1)
```

```
            If inrect.GetValue(1, 1) < 0 And inrect.GetValue(1, 2) > 0 Then
               inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) / inrect.GetValue(1, 1))
* 180 / 3.14159265358979 + 180 + ypolar.GetValue(1, 2)
            ElseIf inrect.GetValue(1, 1) < 0 And inrect.GetValue(1, 2) < 0 Then
                  inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) / inrect.GetValue(1,
1)) * 180 / 3.14159265358979 - 180 + ypolar.GetValue(1, 2)
               Else
                  inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) / inrect.GetValue(1,
1)) * 180 / 3.14159265358979 + ypolar.GetValue(1, 2)
            End If
            ikrect.ChangeValue(1, 1) = vreal.GetValue(K, 1) - vreal.GetValue(N, 1)
            ikrect.ChangeValue(1, 2) = vimg.GetValue(K, 1) - vimg.GetValue(N, 1)
            ikpolar.ChangeValue(1, 1) = Sqr(ikrect.GetValue(1, 1) ^ 2 + ikrect.GetValue(1,
2) ^ 2) * ypolar.GetValue(l, 1)
            If ikrect.GetValue(1, 1) < 0 And ikrect.GetValue(1, 2) > 0 Then
               ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) / ikrect.GetValue(1, 1))
* 180 / 3.14159265358979 + ypolar.GetValue(1, 2) + 180
            ElseIf ikrect.GetValue(1, 1) < 0 And ikrect.GetValue(1, 2) < 0 Then
                  ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) / ikrect.GetValue(1,
1)) * 180 / 3.14159265358979 + ypolar.GetValue(1, 2) - 180
               Else
                  ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) / ikrect.GetValue(1,
1)) * 180 / 3.14159265358979 + ypolar.GetValue(1, 2)
            End If
            snkpolar.ChangeValue(1, 1) = vrealpolar.GetValue(N, 1) * inpolar.GetValue(1, 1)
* 100
            snkpolar.ChangeValue(1, 2) = vangpolar.GetValue(N, 1) - inpolar.GetValue(1, 2)
            sknpolar.ChangeValue(1, 1) = vrealpolar.GetValue(K, 1) * ikpolar.GetValue(1, 1)
* 100
            sknpolar.ChangeValue(1, 2) = vangpolar.GetValue(K, 1) - ikpolar.GetValue(1, 2)
            snkrect.ChangeValue(1, 1) = snkpolar.GetValue(1, 1) *
Cos(snkpolar.GetValue(1, 2) * 3.14159265358979 / 180)
            snkrect.ChangeValue(1, 2) = snkpolar.GetValue(1, 1) * Sin(snkpolar.GetValue(1,
2) * 3.14159265358979 / 180)
            sknrect.ChangeValue(1, 1) = sknpolar.GetValue(1, 1) *
Cos(sknpolar.GetValue(1, 2) * 3.14159265358979 / 180)
            sknrect.ChangeValue(1, 2) = sknpolar.GetValue(1, 1) * Sin(sknpolar.GetValue(1,
2) * 3.14159265358979 / 180)
            slrect.ChangeValue(1, 1) = snkrect.GetValue(1, 1) + sknrect.GetValue(1, 1)
            slrect.ChangeValue(1, 2) = snkrect.GetValue(1, 2) + sknrect.GetValue(1, 2)
            sltrect.ChangeValue(1, 1) = slrect.GetValue(1, 1) + sltrect.GetValue(1, 1)
            sltrect.ChangeValue(1, 2) = slrect.GetValue(1, 2) + sltrect.GetValue(1, 2)
         ElseIf linedatamatrix.GetValue(l, 2) = N Then
               K = linedatamatrix.GetValue(l, 1)
               inrect.ChangeValue(1, 1) = vreal.GetValue(N, 1) - vreal.GetValue(K, 1)
               inrect.ChangeValue(1, 2) = vimg.GetValue(N, 1) - vimg.GetValue(K, 1)
               inpolar.ChangeValue(1, 1) = Sqr(inrect.GetValue(1, 1) ^ 2 +
inrect.GetValue(1, 2) ^ 2) * ypolar.GetValue(l, 1)
               If inrect.GetValue(1, 1) < 0 And inrect.GetValue(1, 2) > 0 Then
                  inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) / inrect.GetValue(1,
1)) * 180 / 3.14159265358979 + 180 + ypolar.GetValue(1, 2)
               ElseIf inrect.GetValue(1, 1) < 0 And inrect.GetValue(1, 2) < 0 Then
                     inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) /
inrect.GetValue(1, 1)) * 180 / 3.14159265358979 - 180 + ypolar.GetValue(1, 2)
                  Else
                     inpolar.ChangeValue(1, 2) = Atn(inrect.GetValue(1, 2) /
inrect.GetValue(1, 1)) * 180 / 3.14159265358979 + ypolar.GetValue(1, 2)
               End If
               ikrect.ChangeValue(1, 1) = vreal.GetValue(K, 1) - vreal.GetValue(N, 1)
```

```
                ikrect.ChangeValue(1, 2) = vimg.GetValue(K, 1) - vimg.GetValue(N, 1)
                ikpolar.ChangeValue(1, 1) = Sqr(ikrect.GetValue(1, 1) ^ 2 +
ikrect.GetValue(1, 2) ^ 2) * ypolar.GetValue(l, 1)
                If ikrect.GetValue(1, 1) < 0 And ikrect.GetValue(1, 2) > 0 Then
                    ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) / ikrect.GetValue(1,
1)) * 180 / 3.14159265358979 + 180 + ypolar.GetValue(1, 2)
                ElseIf ikrect.GetValue(1, 1) < 0 And ikrect.GetValue(1, 2) < 0 Then
                        ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) /
ikrect.GetValue(1, 1)) * 180 / 3.14159265358979 - 180 + ypolar.GetValue(1, 2)
                    Else
                        ikpolar.ChangeValue(1, 2) = Atn(ikrect.GetValue(1, 2) /
ikrect.GetValue(1, 1)) * 180 / 3.14159265358979 + ypolar.GetValue(1, 2)
                End If
                snkpolar.ChangeValue(1, 1) = vrealpolar.GetValue(N, 1) * inpolar.GetValue(1,
1) * 100
                snkpolar.ChangeValue(1, 2) = vangpolar.GetValue(N, 1) - inpolar.GetValue(1,
2)
                sknpolar.ChangeValue(1, 1) = vrealpolar.GetValue(K, 1) * ikpolar.GetValue(1,
1) * 100
                sknpolar.ChangeValue(1, 2) = vangpolar.GetValue(K, 1) - ikpolar.GetValue(1,
2)
                snkrect.ChangeValue(1, 1) = snkpolar.GetValue(1, 1) *
Cos(snkpolar.GetValue(1, 2) * 3.14159265358979 / 180)
                snkrect.ChangeValue(1, 2) = snkpolar.GetValue(1, 1) *
Sin(snkpolar.GetValue(1, 2) * 3.14159265358979 / 180)
                sknrect.ChangeValue(1, 1) = sknpolar.GetValue(1, 1) *
Cos(sknpolar.GetValue(1, 2) * 3.14159265358979 / 180)
                sknrect.ChangeValue(1, 2) = sknpolar.GetValue(1, 1) *
Sin(sknpolar.GetValue(1, 2) * 3.14159265358979 / 180)
                slrect.ChangeValue(1, 1) = snkrect.GetValue(1, 1) + sknrect.GetValue(1, 1)
                slrect.ChangeValue(1, 2) = snkrect.GetValue(1, 2) + sknrect.GetValue(1, 2)
                sltrect.ChangeValue(1, 1) = slrect.GetValue(1, 1) + sltrect.GetValue(1, 1)
                sltrect.ChangeValue(1, 2) = slrect.GetValue(1, 2) + sltrect.GetValue(1, 2)
            End If
        If snkrect.GetValue(1, 1) > 0 Then
            If linedatamatrix.GetValue(l, 1) = N Or linedatamatrix.GetValue(l, 2) = N Then
                Load LALPowerFlow(l - 1)
                If snkrect.GetValue(1, 2) < 0 Then
                    LALPowerFlow(l - 1) = "PowerFlow = " & Round(snkrect.GetValue(1, 1) / 100,
2) & " - " & Round(Abs(snkrect.GetValue(1, 2)) / 100, 2) & " j"
                Else
                    LALPowerFlow(l - 1) = "PowerFlow = " & Round(snkrect.GetValue(1, 1) / 100,
2) & " + " & Round(Abs(snkrect.GetValue(1, 2)) / 100, 2) & " j"
                End If
                Load LALlineloss(l - 1)
                LALlineloss(l - 1) = "LineLoss = " & Round(slrect.GetValue(1, 1) / 100, 4) & " +
" & Round(slrect.GetValue(1, 2) / 100, 4) & " j"
                transnum = linedatamatrix.GetValue(l, 7)
                LALPowerFlow(l - 1).Move LALLineTransmission(transnum - 1).Left +
LALLineTransmission(transnum - 1).Width * 0.5 - LALPowerFlow(l - 1).Width * 0.5,
LALLineTransmission(transnum - 1).Top + LALPowerFlow(l - 1).Height - 40
                LALPowerFlow(l - 1).Visible = True
                LALlineloss(l - 1).Move LALLineTransmission(transnum - 1).Left +
LALLineTransmission(transnum - 1).Width * 0.5 - LALlineloss(l - 1).Width * 0.5,
LALLineTransmission(transnum - 1).Top + LALlineloss(l - 1).Height + 150
                LALlineloss(l - 1).Visible = True
                If linedatamatrix.GetValue(l, 1) < linedatamatrix.GetValue(l, 2) Then
                    AnimationTLS(l - 1) = 1
                Else
```

```
                AnimationTLS(l - 1) = -1
            End If
        End If
    End If
    Next 'this next is meant for L
Next 'this next is meant for n

Dim M As Integer
Dim num As Integer
  num = 0
For M = 0 To ldGenerator.Total
    If ldGenerator.IsExist(M + 1) Then
        Load LALGen(num)
        LALGen(num) = (sm.Power_Generator(LALGenerator(M)) / 100) & "PU"
        LALGen(num).Move LALGenerator(M).Left + LALGenerator(M).Width * 0.5 -
LALGen(num).Width * 0.5, LALGenerator(M).Top + LALGen(num).Height - 40
        LALGen(num).Visible = True
        num = num + 1
    End If
Next
    num = 0
For M = 0 To ldFeeder.Total
    If ldFeeder.IsExist(M + 1) Then
        Load Lalfee(num)
        Lalfee(num) = (sm.Power_Feeder(LALFeeder(M)) / 100) & "+" &
(sm.Reactive_Feeder(LALFeeder(M)) / 100) & "j PU"
        Lalfee(num).Move LALFeeder(M).Left + LALFeeder(M).Width * 0.5 -
Lalfee(num).Width * 0.5, LALFeeder(M).Top + Lalfee(num).Height - 40
        Lalfee(num).Visible = True
        num = num + 1
    End If
Next
    num = 0
For M = 0 To ldLineBusbar.Total
    If ldLineBusbar.IsExist(M + 1) Then
        Load Lallinebus(num)
        If q.GetValue(M + 1, 1) < 0 Then
            Lallinebus(num) = Round(p.GetValue(M + 1, 1), 2) & Round(q.GetValue(M + 1, 1),
2) & "j PU"
        Else
            Lallinebus(num) = Round(p.GetValue(M + 1, 1), 2) & "+" & Round(q.GetValue(M +
1, 1), 2) & "j PU"
        End If
        Lallinebus(num).Move LALLineBusbar(M).Left + LALLineBusbar(M).Width + 0.5 -
Lallinebus(num).Width * 0.5, LALLineBusbar(M).Top + Lallinebus(num).Height - 40
        Lallinebus(num).Visible = True
        num = num + 1
    End If
Next

    TimerSimulationFlow.Enabled = True
End Sub

Public Sub Simulate_LoadFlow(Optional showresult As Boolean = True)
On Error Resume Next
'sm.Tran_Line_Power
    If N_to_N_Detection(False) = False Then 'Checking Node to Node Detection and Creating
Graph Table
```

```
    MsgBox "Please verify the node to node detection first!" & vbNewLine & " Then try again!", vbInformation
    Exit Sub
  End If
  Dim matrixa As New Matrix
  Dim matrixb As New Matrix
  If ldLineBusbar.TotalLoaded < 2 Then
    MsgBox "Load Flow simulations can be performed on more than one busbars", vbInformation
    Exit Sub
  End If
  matrixa.Order ldLineBusbar.TotalLoaded - 1, ldLineBusbar.TotalLoaded - 1
  matrixb.Order ldLineBusbar.TotalLoaded - 1, 1
  '"MatrixA.Order ldLineBusbar.TotalLoaded, ldLineBusbar.TotalLoaded
  '"MatrixB.Order ldLineBusbar.TotalLoaded, 1
  frmWait.Show
  frmWait.lalTotal = matrixb.TotalRows * matrixb.TotalRows + 5
  frmWait.lalCounter = 0
  frmWait.Refresh
  Dim sm As New Simulation
  sm.Initialize DB
  Dim I As Integer, J As Integer, rowno As Integer, colno As Integer

  For I = 0 To ldLineTransmission.Total   'IF the transmission line is failed
    If ldLineTransmission.IsExist(I + 1) Then
      If LineTranStart(I).BorderColor = vbRed Then sm.ZeroPower_TransmissionLine LALLineTransmission(I)
    End If
  Next

  rowno = 0
  For I = 0 To ldLineBusbar.Total - 1
  '"For I = 0 To ldLineBusbar.Total
    If ldLineBusbar.IsExist(I + 1) Then
    rowno = rowno + 1
      colno = 0
      If rowno > matrixa.TotalRows Then Exit For
      For J = 0 To ldLineBusbar.Total - 1
      '"For J = 0 To ldLineBusbar.Total
        If ldLineBusbar.IsExist(J + 1) Then
          colno = colno + 1
          If colno > matrixa.TotalCols Then Exit For
          If rowno = colno Then
            Dim K3 As Integer
            For K3 = 0 To ldLineBusbar.Total
              If ldLineBusbar.IsExist(K3 + 1) Then
                matrixa.ChangeValue(rowno, colno) = matrixa.GetValue(rowno, colno) + sm.Tran_Line_Power(LALLineBusbar(I).Caption, LALLineBusbar(K3).Caption)
              End If
            Next
          Else
            matrixa.ChangeValue(rowno, colno) = -1 * sm.Tran_Line_Power(LALLineBusbar(I).Caption, LALLineBusbar(J).Caption)
          End If
        End If
      Next
    matrixb.ChangeValue(rowno, 1) = sm.BusBar_Net_Power(LALLineBusbar(I).Caption)
    End If
  Next
```

```
    'Dim msg As String
    'For I = 1 To MatrixA.TotalRows
    '    For J = 1 To MatrixA.TotalCols
    '        msg = msg & " " & MatrixA.GetValue(I, J)
    '    Next
    '    msg = msg & " | " & MatrixB.GetValue(I, 1) & vbNewLine
    'Next
    'MsgBox msg
    Dim c As New Matrix, mode As Double
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    mode = matrixa.Determinent
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    If mode = 0 Then
        frmWait.lalCounter = frmWait.lalTotal
        MsgBox "Unique solution of the system is not possible"
        Exit Sub
    End If
    'Set C = MatrixA.AdjointMatrix
    Set c = matrixa.invmatrix
    Dim D As New Matrix
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    Set D = MatMultiply(c, matrixb)
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmSimulationLoadFlowResult.MSFlexGrid1.Rows = D.TotalRows + 1
    frmSimulationLoadFlowResult.MSFlexGrid1.Cols = 2
    frmSimulationLoadFlowResult.MSFlexGrid1.Row = 0
    frmSimulationLoadFlowResult.MSFlexGrid1.col = 0
    frmSimulationLoadFlowResult.MSFlexGrid1.Text = "Result of " & Caption
    frmSimulationLoadFlowResult.MSFlexGrid1.col = 1
    frmSimulationLoadFlowResult.MSFlexGrid1.Text = "PF"
    For I = 1 To D.TotalRows
        For J = 1 To D.TotalCols
            '"D.ChangeValue(i, j) = D.GetValue(i, j) / Mode
            frmSimulationLoadFlowResult.MSFlexGrid1.Row = I
            frmSimulationLoadFlowResult.MSFlexGrid1.col = 0
            frmSimulationLoadFlowResult.MSFlexGrid1.Text = "Phase" & I - 1 & " = " &
D.GetValue(I, J)
            frmSimulationLoadFlowResult.MSFlexGrid1.col = 1
            frmSimulationLoadFlowResult.MSFlexGrid1.Text = Format(Cos(D.GetValue(I, J)),
"0.00000")
        Next
    Next
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    frmSimulationLoadFlowResult.ShowDisplaySimulations matrixa, matrixb
    frmSimulationLoadFlowResult.Caption = "Load Flow Calculation Results [" & Caption &
"]"
    frmSimulationLoadFlowResult.Move MainWindow.ScaleWidth / 2 -
frmSimulationLoadFlowResult.Width / 2, MainWindow.Height / 2 -
frmSimulationLoadFlowResult.Height / 2
    frmSimulationLoadFlowResult.MSFlexGrid1.Width =
frmSimulationLoadFlowResult.Command1.Left -
frmSimulationLoadFlowResult.MSFlexGrid1.Left * 2
    frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(0) =
frmSimulationLoadFlowResult.MSFlexGrid1.Width * 0.73
```

```
    frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(1)                                    =
frmSimulationLoadFlowResult.MSFlexGrid1.Width * 0.22
    If      frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(0)      <      2200      Then
frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(0) = 2200
    If      frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(1)      <      700      Then
frmSimulationLoadFlowResult.MSFlexGrid1.ColWidth(1) = 700
    J = 0
    On Error Resume Next
    For I = 0 To ldGenerator.Total
        If ldGenerator.IsExist(I + 1) Then
            Load LALGen(J)
            LALGen(J) = sm.Power_Generator(LALGenerator(I)) & "PU"
            LALGen(J).Move  LALGenerator(I).Left  +  LALGenerator(I).Width  *  0.5  -
LALGen(J).Width * 0.5, LALGenerator(I).Top + LALGen(J).Height - 40
            LALGen(J).Visible = True
            J = J + 1
        End If
    Next
    J = 0
    For I = 0 To ldFeeder.Total
        If ldFeeder.IsExist(I + 1) Then
            Load Lalfee(J)
            Lalfee(J) = sm.Power_Feeder(LALFeeder(I)) & "PU"
            Lalfee(J).Move  LALFeeder(I).Left  +  LALFeeder(I).Width  *  0.5  -  Lalfee(J).Width  *
0.5, LALFeeder(I).Top + Lalfee(J).Height - 40
            Lalfee(J).Visible = True
            J = J + 1
        End If
    Next
    J = 0
    For I = 0 To ldtransformer.Total
        If ldtransformer.IsExist(I + 1) Then
            Load LalInfo(J)
            LalInfo(J) = sm.Power_Transformer(LALTransformer(I)) & "PU"
            LalInfo(J).Move  LALTransformer(I).Left  +  LALTransformer(I).Width  *  0.5  -
LalInfo(J).Width * 0.5, LALTransformer(I).Top + LalInfo(J).Height - 40
            LalInfo(J).Visible = True
            J = J + 1
        End If
    Next
    'For I = 0 To ldCircuitBreaker.Total
    '    If ldCircuitBreaker.IsExist(I + 1) Then
    '        J = J + 1
    '        Load LalInfo(J)
    '        LalInfo(J) = Sm.Power_Circuitbreaker(LALCircuitbreaker(I)) & "PU"
    '            LalInfo(J).Move  LALCircuitbreaker(I).Left  +  LALCircuitbreaker(I).Width  *  0.5  -
LalInfo(J).Width * 0.5, LALCircuitbreaker(I).Top + LalInfo(J).Height - 40
    '        LalInfo(J).Visible = True
    '    End If
    'Next
    J = 0
    For I = 0 To ldLineTransmission.Total
        If ldLineTransmission.IsExist(I + 1) Then
                Load LalInfo(J)
                Load LALlineloss(J)
                LalInfo(J) = sm.Power_TransmissionLine(LALLineTransmission(I)) & "PU"
                LalInfo(J).Move  LALLineTransmission(I).Left  +  LALLineTransmission(I).Width  *
0.5 - LalInfo(J).Width * 0.5, LALLineTransmission(I).Top + LalInfo(J).Height - 40
                LalInfo(J).Visible = True
```

```
            If ldLineBusbar.Total = 2 Then
                'The Change 9.1.1
                LalInfo(J) = LalInfo(J) & "  PF=" & Format((D.GetValue(1, 1) - D.GetValue(2, 1))
* sm.Power_TransmissionLine(LALLineTransmission(I)), "#.00##")
            End If
                LALlineloss(J) = 0
                J = J + 1
        End If
    Next
    Dim TempB() As String
    ReDim TempB(Int(D.TotalRows))
    For I = 1 To D.TotalRows
        TempB(I - 1) = "Phase" & I & " = " & Format(D.GetValue(I, 1), "0.0000") '&
vbNewLine & "PF = " & Format(Cos(D.GetValue(I, 1)), "0.0000")
    Next
    Dim K As Integer
    K = 0
    J = 0
    For I = 0 To ldLineBusbar.Total
        If ldLineBusbar.IsExist(I + 1) Then
            Load Lallinebus(J)
            Lallinebus(J) = TempB(K)
            Lallinebus(J).Move LALLineBusbar(I).Left + LALLineBusbar(I).Width * 0.5 -
Lallinebus(J).Width * 0.5, LALLineBusbar(I).Top + Lallinebus(J).Height - 40
            Lallinebus(J).Visible = True
            K = K + 1
            J = J + 1
        End If
    Next

    frmSimulationLoadFlowResult.Show

End Sub


Private Sub Form_Resize()
    On Error GoTo Finish
    If WindowState <> 1 Then
        VScroll.Move ScaleWidth + ScaleLeft - VScroll.Width, 0, VScroll.Width, ScaleHeight +
ScaleTop
        HScroll.Move 0, ScaleHeight + ScaleTop - HScroll.Height, ScaleWidth + ScaleLeft -
VScroll.Width
        HScroll.Min = 0
        VScroll.Min = 0
        If PicDrawing.Width - ScaleWidth + HScroll.Height + 360 > 0 Then
            HScroll.Visible = True
            HScroll.Max = Abs(PicDrawing.Width - ScaleWidth + HScroll.Height + 360) / 100
        Else
            HScroll.Value = 0
            HScroll.Visible = False
        End If
        If PicDrawing.Height - ScaleHeight + HScroll.Height + 360 > 0 Then
            VScroll.Visible = True
            VScroll.Max = Abs(PicDrawing.Height - ScaleHeight + HScroll.Height + 360) / 100
        Else
            VScroll.Value = 0
            VScroll.Visible = False
        End If
        PicScaleHorizental.Width = HScroll.Width + VScroll.Width
```

```
        PicScaleVertical.Height = VScroll.Height
        Call HScroll_Change
        Call VScroll_Change
    End If
Finish:

End Sub

Private Sub LalInfo_DragOver(Index As Integer, Source As Control, X As Single, Y As Single,
State As Integer)
    Call PicDrawing_DragDrop(Source, LalInfo(Index).Left + X, LalInfo(Index).Top + Y)
End Sub

Private Sub PicDrawing_DragDrop(Source As Control, X As Single, Y As Single)
    Source.Left = X - Source.Width / 2
    Source.Top = Y - Source.Height / 2
End Sub

Private Sub LALGenerator_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
    Me.LALGenerator(Index).Move X - Me.LALGenerator(Index).Width / 2, Y -
Me.LALGenerator(Index).Height / 2
    Me.LALGenerator(Index).Visible = False
End Sub

Private Sub LALGenerator_Mouseup(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
    Me.LALGenerator(Index).Visible = True
    Me.LALGenerator(Index).Move X - Me.LALGenerator(Index).Width / 2, Y -
Me.LALGenerator(Index).Height / 2
    CurrentControlID = LALGenerator(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALGen_MouseDown(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
    Me.LALGen(Index).Move X - Me.LALGen(Index).Width / 2, Y - Me.LALGen(Index).Height
/ 2
    Me.LALGen(Index).Visible = False
End Sub

Private Sub LALGen_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As
Single, Y As Single)
    Me.LALGen(Index).Visible = True
    Me.LALGen(Index).Move X - Me.LALGen(Index).Width / 2, Y - Me.LALGen(Index).Height
/ 2
    CurrentControlID = LALGen(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALCircuitBreaker_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
    Me.LALCircuitbreaker(Index).Move X - Me.LALCircuitbreaker(Index).Width / 2, Y -
LALCircuitbreaker(Index).Height / 2
    Me.LALCircuitbreaker(Index).Visible = False
End Sub

Private Sub LALCircuitBreaker_Mouseup(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
```

```
    Me.LALCircuitbreaker(Index).Visible = True
    Me.LALCircuitbreaker(Index).Move X - Me.LALCircuitbreaker(Index).Width / 2, Y -
LALCircuitbreaker(Index).Height / 2
    CurrentControlID = LALCircuitbreaker(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALpowerflow_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
    Me.LALPowerFlow(Index).Move X - Me.LALPowerFlow(Index).Width / 2, Y -
LALPowerFlow(Index).Height / 2
    Me.LALPowerFlow(Index).Visible = False
End Sub

Private Sub LALpowerflow_Mouseup(Index As Integer, button As Integer, Shift As Integer,
X As Single, Y As Single)
    Me.LALPowerFlow(Index).Visible = True
    Me.LALPowerFlow(Index).Move X - Me.LALPowerFlow(Index).Width / 2, Y -
LALPowerFlow(Index).Height / 2
    CurrentControlID = LALPowerFlow(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALlineloss_MouseDown(Index As Integer, button As Integer, Shift As Integer,
X As Single, Y As Single)
    Me.LALlineloss(Index).Move X - Me.LALlineloss(Index).Width / 2, Y -
LALlineloss(Index).Height / 2
    Me.LALlineloss(Index).Visible = False
End Sub

Private Sub LALlineloss_Mouseup(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
    Me.LALlineloss(Index).Visible = True
    Me.LALlineloss(Index).Move X - Me.LALlineloss(Index).Width / 2, Y -
LALlineloss(Index).Height / 2
    CurrentControlID = LALlineloss(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALfeeder_MouseDown(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
    Me.LALFeeder(Index).Move X - Me.LALFeeder(Index).Width / 2, Y -
Me.LALFeeder(Index).Height / 2
    Me.LALFeeder(Index).Visible = False
End Sub

Private Sub LALfeeder_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As
Single, Y As Single)
    Me.LALFeeder(Index).Visible = True
    Me.LALFeeder(Index).Move X - Me.LALFeeder(Index).Width / 2, Y -
Me.LALFeeder(Index).Height / 2
    CurrentControlID = LALFeeder(Index)
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALfee_MouseDown(Index As Integer, button As Integer, Shift As Integer, X As
Single, Y As Single)
    Me.Lalfee(Index).Move X - Me.Lalfee(Index).Width / 2, Y - Me.Lalfee(Index).Height / 2
    Me.Lalfee(Index).Visible = False
```

```
End Sub

Private Sub LALfee_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As
Single, Y As Single)
   Me.Lalfee(Index).Visible = True
   Me.Lalfee(Index).Move X - Me.Lalfee(Index).Width / 2, Y - Me.Lalfee(Index).Height / 2
   CurrentControlID = Lalfee(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALlinebusbar_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   Me.LALLineBusbar(Index).Move X - Me.LALLineBusbar(Index).Width / 2, Y -
Me.LALLineBusbar(Index).Height / 2
   Me.LALLineBusbar(Index).Visible = False
End Sub

Private Sub LALlinebusbar_Mouseup(Index As Integer, button As Integer, Shift As Integer,
X As Single, Y As Single)
   Me.LALLineBusbar(Index).Visible = True
   Me.LALLineBusbar(Index).Move X - Me.LALLineBusbar(Index).Width / 2, Y -
Me.LALLineBusbar(Index).Height / 2
   CurrentControlID = LALLineBusbar(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALlinebus_MouseDown(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
   Me.Lallinebus(Index).Move X - Me.Lallinebus(Index).Width / 2, Y -
Me.Lallinebus(Index).Height / 2
   Me.Lallinebus(Index).Visible = False
End Sub

Private Sub LALlinebus_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As
Single, Y As Single)
   Me.Lallinebus(Index).Visible = True
   Me.Lallinebus(Index).Move X - Me.Lallinebus(Index).Width / 2, Y -
Me.Lallinebus(Index).Height / 2
   CurrentControlID = Lallinebus(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALlinetransmission_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   Me.LALLineTransmission(Index).Move X - Me.LALLineTransmission(Index).Width / 2, Y -
Me.LALLineTransmission(Index).Height / 2
   Me.LALLineTransmission(Index).Visible = False
End Sub

Private Sub LALlinetransmission_Mouseup(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   Me.LALLineTransmission(Index).Visible = True
   Me.LALLineTransmission(Index).Move X - Me.LALLineTransmission(Index).Width / 2, Y -
Me.LALLineTransmission(Index).Height / 2
   CurrentControlID = LALLineTransmission(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub
```

```
Private Sub LALtransformer_MouseDown(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
   Me.LALTransformer(Index).Move X - Me.LALTransformer(Index).Width / 2, Y - Me.LALTransformer(Index).Height / 2
   Me.LALTransformer(Index).Visible = False
End Sub

Private Sub LALtransformer_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
   Me.LALTransformer(Index).Visible = True
   Me.LALTransformer(Index).Move X - Me.LALTransformer(Index).Width / 2, Y - Me.LALTransformer(Index).Height / 2
   CurrentControlID = LALTransformer(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub LALinfo_MouseDown(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
   Me.LalInfo(Index).Move X - Me.LalInfo(Index).Width / 2, Y - Me.LalInfo(Index).Height / 2
   Me.LalInfo(Index).Visible = False
End Sub

Private Sub LALinfo_Mouseup(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
   Me.LalInfo(Index).Visible = True
   Me.LalInfo(Index).Move X - Me.LalInfo(Index).Width / 2, Y - Me.LalInfo(Index).Height / 2
   CurrentControlID = LalInfo(Index)
   OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
End Sub

Private Sub VScroll_Change()

   If VScroll.Value = VScroll.Max Then
      PicDrawing.Top = -VScroll.Max * 100
   ElseIf VScroll.Value = VScroll.Min Then
      PicDrawing.Top = 0
   Else
      PicDrawing.Top = -VScroll.Value * 100
   End If
   Dim I As Integer
   PicScaleVertical.Cls
   For I = PicDrawing.Top To Width Step 144
      If (I - PicDrawing.Top) Mod 5 = 0 Then
         PicScaleVertical.Line (0, I)-(PicScaleVertical.Width / 2, I)
      Else
         PicScaleVertical.Line (0, I)-(PicScaleVertical.Width / 4, I)
      End If
   Next
   For I = PicDrawing.Top To Width Step 1440
      PicScaleVertical.CurrentX = 0
      PicScaleVertical.CurrentY = I 'PicScaleVertical.Height / 3
      PicScaleVertical.Print Int((I - PicDrawing.Top) / 1440)
      PicScaleVertical.Line (0, I)-(PicScaleVertical.Width, I), RGB(200, 50, 50)
   Next
End Sub

Private Sub VScroll_Scroll()
```

```
    Call VScroll_Change
End Sub

Private Sub HScroll_Change()

    If HScroll.Value = HScroll.Max Then
        PicDrawing.Left = -HScroll.Max * 100
    ElseIf HScroll.Value = HScroll.Min Then
        PicDrawing.Left = 0
    Else
        PicDrawing.Left = -HScroll.Value * 100
    End If
    Dim I As Integer
    PicScaleHorizental.Cls
    For I = PicDrawing.Left To Width Step 144
        If (I - PicDrawing.Left) Mod 5 = 0 Then
            PicScaleHorizental.Line (I, 0)-(I, PicScaleHorizental.Height / 2)
        Else
            PicScaleHorizental.Line (I, 0)-(I, PicScaleHorizental.Height / 4)
        End If
    Next
    For I = PicDrawing.Left To Width Step 1440
        PicScaleHorizental.CurrentX = I
        PicScaleHorizental.CurrentY = 0 'PicScaleHorizental.Height / 3
        PicScaleHorizental.Print Int((I - PicDrawing.Left) / 1440)
        PicScaleHorizental.Line (I, 0)-(I, PicScaleHorizental.Height), RGB(200, 50, 50)
    Next
End Sub

Private Sub HScroll_Scroll()
    Call HScroll_Change
End Sub

Private Sub Form_Load()
    Hide
    DoEvents
    SaveMe = False   'By default a model is already saved
    ReDim BlinkLAL(0)   'initializing
    Dim H As Integer, W As Integer, I, J
    '----temprarily storing forms dimensions------
    H = Height
    W = Width
    '--------------------

    PicDrawing.Move 0, 0

    'Making form of the maxium area
    Width = Screen.Width
    Height = Screen.Height


    'Opening Database "NewFileName". It is already exists. Because if it is not exists
    'Form cannot be loaded. It is a communication variable between MainWindow as form
    Set DB = OpenDatabase(AppPath & NewFileName)

    'Shall not make NewFileName empty( NewFileName="" ) because it is used for
    'further processing
    'If Opendimension = True Then GetDimension Me, PicDrawing, DB
```

```
   'Restoring forms dimensions
   DrawWidth = 2
   Height = H
   Width = W
End Sub 'Form Load


Private Sub Form_Activate()
   'WHEN EVER USER SELECTS THIS FORM WINDOW IT UPDATES THE COMMON
VARIABLE
   'SO THAT REST OF THE APPLICATION CAN COMMUNICATE WITH THIS FORM
   ActiveFormID = MyID
   EnableAll True
End Sub

Private Sub PicDrawing_MouseDown(button As Integer, Shift As Integer, X As Single, Y As
Single)
'Here shall draw all the objects
If ButtonKey <> "" Then
   MousePointer = 2
   ResetAllFlags
   SaveMe = True   'Because a change is happened, So shall save form
'    NewItem = True
   Select Case LCase(ButtonKey)    'THIS BUTTONKEY IS COMMUNICATION VARIABLE
                              'BETWEEN FORM(ME) AND MAINWINDOW'S TOOLBAR

   'ALL THE CASES ARE SIMILAR TO THE FIRST CASE SO READ IT CAREFULLY.
   'REST OF CAESE ARE NOT COMMENTED
   Case Is = "transmissionline"   'Draw Transmission line

      Current = ldLineTransmission.NewItemNumber  'GETTING AN INTEGER OF WHICH
                                    'INDEXED TRANSMISSION LINE IS NOT
                                    'CREATED OR DELETED BY THE USER
                                    'HENCE LINE WITH THIS INDEX NOT EXISTS
      If Current > 0 Then 'HAVE TRANSMISSION LINE WITH INDEX ZERO AT DEGIN TIME
SO
                     'SHALL NOT LOAD(CREATE) IT.

         Load LineTranStart(Current)  'CREATING(LOADING) NEW TRANSMISSION LINE
         Load LineTranMid(Current) 'SINCE TRANSMISSION LINE CONSIST OF Three LINES
         Load LineTranEnd(Current) 'SINCE TRANSMISSION LINE CONSIST OF Three LINES
         Load LALLineTransmission(Current)    'LABEL(TO DISPLAY ID) OF TRANSMISSION
LINE
               'REMEMBER ALL THE LABELS AND INTERRELATED TRANSMISSION LINES
HAVE SAME INDEX
               'SO TO GET ID OF THE SELECTED OBJECT, I USED IT'S ASSOCIATED LABEL
         bTransmissionLineBend1.OpenNewAtIndex Current + 1
         bTransmissionLineBend2.OpenNewAtIndex Current + 1

      End If

      'BY THE FOLLOWING STATEMENTS I MAKE THE LINE EXACTLY AT THE POINT OF
MOUSE POINTER
      LineTranStart(Current).X1 = X     'SETTING X1 OF LINE TO THE X-COORDINATE OF
THE MOUSE
      LineTranStart(Current).Y1 = Y     'SETTING Y1 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranStart(Current).X2 = X     'SETTING X2 OF LINE TO THE X-COORDINATE OF
THE MOUSE
```

```
      LineTranStart(Current).Y2 = Y     'SETTING Y2 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranStart(Current).Visible = True 'ALL THE LINES ARE NOT VISIBLE, SO I AM
MAKING IT SO
      LineTranMid(Current).X1 = X    'WSETTING X1 OF LINE TO THE X-COORDINATE OF
THE MOUSE
      LineTranMid(Current).Y1 = Y     'SETTING Y1 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranMid(Current).X2 = X     'SETTING X2 OF LINE TO THE X-COORDINATE OF
THE MOUSE
      LineTranMid(Current).Y2 = Y     'SETTING Y2 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranMid(Current).Visible = True 'ALL THE LINES ARE NOT VISIBLE, SO I AM
MAKING IT SO
      LineTranEnd(Current).X1 = X     'SETTING X1 OF LINE TO THE X-COORDINATE OF
THE MOUSE
      LineTranEnd(Current).Y1 = Y     'SETTING Y1 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranEnd(Current).X2 = X     'SETTING X2 OF LINE TO THE X-COORDINATE OF
THE MOUSE
      LineTranEnd(Current).Y2 = Y     'SETTING Y2 OF LINE TO THE Y-COORDINATE OF
THE MOUSE
      LineTranEnd(Current).Visible = True 'ALL THE LINES ARE NOT VISIBLE, SO MAKING
IT SO

      LALLineTransmission(Current) = "TL" & Current + 1     'CREATING NEW ID FOR
TRANSMISSION LINE
                    'HERE I USED Current + 1 -->> WHY NOT USE Current AS PART OF ID?
                    'SINCE CONTROL IDs STARTS FORM 0 AND TO STRAT FROM 1
      LALLineTransmission(Current).Move X, Y   'MOVING THIS LABEL(WHICH CANTAINS
LINE ID) EXATLY
                                    'AT THE POSITION OF THE LINE
      LALLineTransmission(Current).Visible = True 'ALL THE LABELS ARE NOT VISIBLE, SO
MAKING IT SO

      CurrentControlID = LALLineTransmission(Current) 'SINCE "CurrentControlID" USED
            'THROUGHTOUT FORM FOR OTHER PURPUSES. E.G. MOVING A CONTROL,
            'GETTING INFORMATION OF CONTROL, DELETING A CONTROL

      EditLineTransmission = True 'NOW EDITING TRANSMISSION LINE BY
                        'MOUSE MOVE OPERATION
      bTransmissionLineBend1.OpenNewAtIndex Current + 1 'DO NOT WANT A BEND IN
TRASMISSION LINE AR NOW
      bTransmissionLineBend2.OpenNewAtIndex Current + 1
   Case Is = "busbar"   'Draw Busbar line
      Current = ldLineBusbar.NewItemNumber    'GETING INTEGER
      If Current > 0 Then     'CONTROL EXIST WITH INDEX ZERO
         Load LineBusbar(Current)    'CREATING NEW CONTROL
         Load LALLineBusbar(Current) 'CREATING NEW LABEL
      End If
      LineBusbar(Current).X1 = X
      LineBusbar(Current).Y1 = Y
      LineBusbar(Current).X2 = X
      LineBusbar(Current).Y2 = Y
      LineBusbar(Current).Visible = True

      LALLineBusbar(Current).Move X, Y
      LALLineBusbar(Current) = "B" & Current + 1
      LALLineBusbar(Current).Visible = True
```

```
        CurrentControlID = LALLineBusbar(Current)
        EditLineBusbar = True

    Case Is = "transformer"
        EditTransformer = True
        Current = ldtransformer.NewItemNumber
        If Current > 0 Then
            Load imgTransformer(Current)
            Load LALTransformer(Current)
        End If
        imgTransformer(Current).Picture = Resource.Transformer(1).Picture
        imgTransformer(Current).Visible = True
        imgTransformer(Current).Move X - imgTransformer(Current).Width / 2, Y -
imgTransformer(Current).Height / 2
        LALTransformer(Current) = "T" & Current + 1
        LALTransformer(Current).Visible = True
        CurrentControlID = LALTransformer(Current)

    Case Is = "generator"
        EditGenerator = True
        Current = ldGenerator.NewItemNumber
        If Current > 0 Then
            Load imgGenerator(Current)
            Load LALGenerator(Current)
        End If
        imgGenerator(Current).Picture = Resource.Generator(1).Picture
        imgGenerator(Current).Visible = True
        imgGenerator(Current).Move X - imgGenerator(Current).Width / 2, Y -
imgGenerator(Current).Height / 2
        LALGenerator(Current) = "G" & Current + 1
        LALGenerator(Current).Visible = True
        CurrentControlID = LALGenerator(Current)

    Case Is = "circuitbreaker"
        EditCuircuitBreaker = True
        Current = ldCircuitBreaker.NewItemNumber
        If Current > 0 Then
            Load imgCircuitbreaker(Current)
            Load LALCircuitbreaker(Current)
        End If
        imgCircuitbreaker(Current).Picture = Resource.Circuitbreaker(1).Picture
        imgCircuitbreaker(Current).Visible = True
        imgCircuitbreaker(Current).Move X - imgCircuitbreaker(Current).Width / 2, Y -
imgCircuitbreaker(Current).Height / 2
        LALCircuitbreaker(Current) = "CB" & Current + 1
        LALCircuitbreaker(Current).Visible = True
        CurrentControlID = LALCircuitbreaker(Current)

    Case Is = "feeder"
         EditFeeder = True
        Current = ldFeeder.NewItemNumber
        If Current > 0 Then
            Load imgFeeder(Current)
            Load LALFeeder(Current)
        End If
        imgFeeder(Current).Picture = Resource.Feeder(1).Picture
        imgFeeder(Current).Visible = True
        imgFeeder(Current).Move X - imgFeeder(Current).Width / 2, Y -
imgFeeder(Current).Height / 2
```

```
        LALFeeder(Current) = "F" & Current + 1
        LALFeeder(Current).Visible = True
        CurrentControlID = LALFeeder(Current)

    End Select
    Editing = True 'Because created componet now I shall edit it
    PicDrawing_MouseMove button, Shift, X, Y 'Methos is called here only to move the
                                'respective label to respective control

Else 'ButtonKey<>""

    DisableAllSquares   'This procedure hides all the selection points of a line
    MousePointer = 0 'Make mouse the normal

End If 'ButtonKey<>""

End Sub 'PicDrawing_MouseDown

Private Sub PicDrawing_MouseMove(button As Integer, Shift As Integer, X As Single, Y As
Single)
    'Me.ToolTipText = "( " & X & " , " & Y & " )"
    If ButtonKey <> "" Then
        MousePointer = 2    'CHANGING MOUSE POINTER TO CROSS
    Else
        MousePointer = 0    'CHANGING IT BACK TO DEFAULT
    End If
    If Editing = True Then  'Moving component when editing is allowed

        'SINCE FOLLOWING INSTRUCTIONS RUNS ONLY WHEN A NEW COMPONET IS
CREATED
        'BECAUSE AFTER CREATION, COMPONETS CAN BE MOVED BY THEIR MouseMove
EVENTS

        If EditLineTransmission = True Then  'TRANSMISSION LINE
            'By the following 2 statments MOVING POINT # 2 TO THE MOUSE X,Y
COORDINATE
            LineTranEnd(Current).X2 = X    'SETTING X2 OF LINE TO THE X-COORDINATE OF
THE MOUSE
            LineTranEnd(Current).Y2 = Y    'SETTING Y2 OF LINE TO THE Y-COORDINATE OF
THE MOUSE

            'FOLLOWING IS THE LOGICAL JION OF THE LINES
            LineTranMid(Current).X1 = (LineTranEnd(Current).X2 + LineTranStart(Current).X1
* 2) / 3
            LineTranMid(Current).X2       =       (LineTranEnd(Current).X2       *       2       +
LineTranStart(Current).X1) / 3
            LineTranMid(Current).Y1 = (LineTranEnd(Current).Y2 + LineTranStart(Current).Y1
* 2) / 3
            LineTranMid(Current).Y2       =       (LineTranEnd(Current).Y2       *       2       +
LineTranStart(Current).Y1) / 3

            LineTranEnd(Current).X1 = LineTranMid(Current).X2
            LineTranEnd(Current).Y1 = LineTranMid(Current).Y2
            LineTranStart(Current).X2 = LineTranMid(Current).X1
            LineTranStart(Current).Y2 = LineTranMid(Current).Y1

            'THE FOLLOWING FUNCTION IS RESPONSIBLE FOR THE POISTION OF THE LINE
LABEL
            AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
```

```
        ElseIf EditLineBusbar = True Then  'BUSBAR LINE
            LineBusbar(Current).X2 = X
            LineBusbar(Current).Y2 = Y
            AdjustLineLabel LineBusbar(Current), LALLineBusbar(Current)
        ElseIf EditTransformer Then
            'HERE MOVING MIDDLE OF COMPNENT(TRANSFORMER) TO THE X,Y OF MOUSE
            imgTransformer(Current).Move X - imgTransformer(Current).Width / 2, Y -
imgTransformer(Current).Height / 2
            LALTransformer(Current).Move imgTransformer(Current).Left +
imgTransformer(Current).Width / 2 - LALTransformer(Current).Width / 2,
imgTransformer(Current).Top + imgTransformer(Current).Height

        ElseIf EditCuircuitBreaker Then
            imgCircuitbreaker(Current).Move X - imgCircuitbreaker(Current).Width / 2, Y -
imgCircuitbreaker(Current).Height / 2
            LALCircuitbreaker(Current).Move imgCircuitbreaker(Current).Left +
imgCircuitbreaker(Current).Width / 2 - LALCircuitbreaker(Current).Width / 2,
imgCircuitbreaker(Current).Top + imgCircuitbreaker(Current).Height
        ElseIf EditGenerator Then
            imgGenerator(Current).Move X - imgGenerator(Current).Width / 2, Y -
imgGenerator(Current).Height / 2
            '"LALGenerator(Current).Move imgGenerator(Current).Left +
imgGenerator(Current).Width / 2 - LALGenerator(Current).Width / 2,
imgGenerator(Current).Top + imgGenerator(Current).Height
        ElseIf EditFeeder Then
            imgFeeder(Current).Move X - imgFeeder(Current).Width / 2, Y -
imgFeeder(Current).Height / 2
            LALFeeder(Current).Move imgFeeder(Current).Left + imgFeeder(Current).Width / 2
- LALFeeder(Current).Width / 2, imgFeeder(Current).Top + imgFeeder(Current).Height
        End If
    End If 'ButtonKey<>""
End Sub 'PicDrawing_MouseMove
'

Private Sub PicDrawing_MouseUp(button As Integer, Shift As Integer, X As Single, Y As
Single)

If ButtonKey <> "" Then 'New component is created

    ButtonKey = "" 'Since created component.So it is
    Editing = False 'Since the newly component is edited once completly now

    'The following procedure opens the chracteristics form of
    'the componet which have id(CurrentControlID) and store it's
    'information in the database(Db)
    'OpenMyInfoForm CurrentControlID, Db, ShowNone
    OpenMyInfoForm CurrentControlID, DB, ShowCharacteristics
    DisableAllButton '<-- This procedure makes the MainWindow Toolbar Normal(All
buttons seems to be unselected)

Else
    DisableAllSquares '<-- This procedure hides all selection points
    ResetAllFlags '<-- This procedure set all flags(EditBusbar ,EditCuircuitBreaker etc) to
falls
    CurrentControlID = "" 'component is selected. Because a useless mouse Up event with
user point of view
    SelectLine X, Y 'Only possiblity is --> User try to select a line
```

```
    If CurrentControlID <> "" And button = 2 Then    'IT MEAN A LINE IS SELECTED AND
USER CLICKS RIGHT BUTTON
        MainWindow.mnu_ApplyFailureCondition.Visible = False
        MainWindow.mnu_RemoveFailureCondition.Visible = False
        If InStr(CurrentControlID, "TL") Then
            Dim Index As Integer
            Index = Right(CurrentControlID, Len(CurrentControlID) - 2)

            If LineTranStart(Index - 1).BorderColor = vbRed Then
                MainWindow.mnu_RemoveFailureCondition.Visible = True
            Else
                MainWindow.mnu_ApplyFailureCondition.Visible = True
            End If
        End If
        PopupMenu MainWindow.mnuControlInfo      'SO SHALL POPUP MENU FOR A LINE
CONTROL
    Else
        Editing = False
    End If
End If

End Sub

Private Sub Form_Unload(Cancel As Integer)
    On Error Resume Next
    If SaveMe = True Then    'MEAN SOME CHANGING IS OCCURED SO NEED TO PROMPT
USER
        Dim response
        response = MsgBox("Do you want to save changes to " & Caption, vbYesNoCancel +
vbExclamation, MainWindow.Caption)
        If response = vbYes Then    'USER AGREE TO SAVE
            Me.SaveFile     'SHALL CALL THE FORM SAVE METHOD
            DB.Close    'Must close database
            'Otherwise cannot open the database again in the single execution of this program
            EnableAll False
        ElseIf response = vbNo Then
            DB.Close    'Must close database
            EnableAll False
        Else
            Cancel = 1  'user response is cancel. So don't close this form
            CloseApp = 0
        End If
    Else 'Don't need to save
        DB.Close    'Must close database
        EnableAll False
    End If

End Sub


'The following code is only for controlling events on Feeder components
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
Private Sub imgFeeder_MouseDown(Index As Integer, button As Integer, Shift As Integer, X
As Single, Y As Single)
    'Sshall move component when mouse is Left clicked
    'Procedure "ImageMouseDown" move the center of an image to the X,Y Coordinates of
mouse
```

```
    'This procedure also make Editing = True to initiate Moving with mouse
    If button = 1 Then ImageMouseDown imgFeeder(Index), X, Y, LALFeeder(Index), ldFeeder.RotationAngle(Index)

End Sub
Private Sub imgFeeder_MouseMove(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    'Shall calling following procedure to move component
    'The following procedure do not move if Editing = False
    ImageMouseMove imgFeeder(Index), X, Y, LALFeeder(Index), ldFeeder.RotationAngle(Index)
End Sub
Private Sub imgFeeder_MouseUp(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    'Shall calling following procedure to move component
    'The following procedure do not move if Editing = False
    'It also update the database according to new position
    ImageMouseUp imgFeeder(Index), X, Y, button, LALFeeder(Index)
End Sub
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

'SAME AS FEEDER
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
Private Sub imgGenerator_MouseDown(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    If button = 1 Then ImageMouseDown imgGenerator(Index), X, Y, LALGenerator(Index), ldGenerator.RotationAngle(Index)
End Sub
Private Sub imgGenerator_MouseMove(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    ImageMouseMove imgGenerator(Index), X, Y, LALGenerator(Index), ldGenerator.RotationAngle(Index)
End Sub
Private Sub imgGenerator_MouseUp(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    ImageMouseUp imgGenerator(Index), X, Y, button, LALGenerator(Index)
End Sub
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

'SAME AS FEEDER
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
Private Sub imgCircuitbreaker_MouseDown(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    If button = 1 Then ImageMouseDown imgCircuitbreaker(Index), X, Y, LALCircuitbreaker(Index), ldCircuitBreaker.RotationAngle(Index)
End Sub
Private Sub imgCircuitbreaker_MouseMove(Index As Integer, button As Integer, Shift As Integer, X As Single, Y As Single)
    ImageMouseMove imgCircuitbreaker(Index), X, Y, LALCircuitbreaker(Index), ldCircuitBreaker.RotationAngle(Index)
End Sub
```

```
Private Sub imgCircuitbreaker_MouseUp(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   ImageMouseUp imgCircuitbreaker(Index), X, Y, button, LALCircuitbreaker(Index)
End Sub
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

'SAME AS FEEDER
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
Private Sub imgtransformer_MouseDown(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   If button = 1 Then ImageMouseDown imgTransformer(Index), X, Y,
LALTransformer(Index), ldtransformer.RotationAngle(Index)
End Sub
Private Sub imgtransformer_MouseMove(Index As Integer, button As Integer, Shift As
Integer, X As Single, Y As Single)
   ImageMouseMove imgTransformer(Index), X, Y, LALTransformer(Index),
ldtransformer.RotationAngle(Index)
End Sub
Private Sub imgtransformer_MouseUp(Index As Integer, button As Integer, Shift As Integer,
X As Single, Y As Single)
   ImageMouseUp imgTransformer(Index), X, Y, button, LALTransformer(Index)
End Sub
'=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

'THE FOLLOWING CODE ENCLOSED BY */*/*/
'IS RESPOSIBLE FOR THE CHANGE OF X1,Y1 POINT OF THE CURRENT LINE.
'IT ALSO DISPLAY A LABEL AS A SQUARE BOX AT X1,Y1 TO GIVE USER RESIZING EFFACT
'*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
/*/*/*/*/*/*/*/
Private Sub RectLinePoint1_MouseDown(button As Integer, Shift As Integer, X As Single, Y
As Single)
   Editing = True
End Sub
Private Sub RectLinePoint1_MouseMove(button As Integer, Shift As Integer, X As Single, Y
As Single)
   If Editing = True Then
      RectLinePoint1.Move RectLinePoint1.Left + X - RectLinePoint1.Width / 2,
RectLinePoint1.Top + Y - RectLinePoint1.Height / 2
      If EditLineBusbar = True Then
         LineBusbar(Current).X1 = RectLinePoint1.Left + RectLinePoint1.Width / 2
         LineBusbar(Current).Y1 = RectLinePoint1.Top + RectLinePoint1.Height / 2
         AdjustLineLabel LineBusbar(Current), LALLineBusbar(Current)
      ElseIf EditLineTransmission = True Then
         If bTransmissionLineBend1.IsExist(Current + 1) = True Then
            LineTranStart(Current).X1 = RectLinePoint1.Left + RectLinePoint1.Width / 2
            LineTranStart(Current).Y1 = RectLinePoint1.Top + RectLinePoint1.Height / 2
         Else
            If bTransmissionLineBend2.IsExist(Current + 1) = True Then
               LineTranStart(Current).X1 = RectLinePoint1.Left + RectLinePoint1.Width / 2
               LineTranStart(Current).Y1 = RectLinePoint1.Top + RectLinePoint1.Height / 2
               LineTranStart(Current).X2 = (LineTranStart(Current).X1 +
LineTranMid(Current).X2) / 2
```

```
                LineTranStart(Current).Y2 = (LineTranStart(Current).Y1 + LineTranMid(Current).Y2) / 2
                LineTranMid(Current).X1 = (LineTranStart(Current).X1 + LineTranMid(Current).X2) / 2
                LineTranMid(Current).Y1 = (LineTranStart(Current).Y1 + LineTranMid(Current).Y2) / 2
                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2, LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
             Else
                LineTranStart(Current).X1 = RectLinePoint1.Left + RectLinePoint1.Width / 2
                LineTranStart(Current).Y1 = RectLinePoint1.Top + RectLinePoint1.Height / 2

                LineTranEnd(Current).X1 = (LineTranStart(Current).X1 + LineTranMid(Current).X2) / 2
                LineTranEnd(Current).Y1 = (LineTranStart(Current).Y1 + LineTranMid(Current).Y2) / 2
                LineTranMid(Current).X1 = (LineTranEnd(Current).X2 + LineTranStart(Current).X1 * 2) / 3
                LineTranMid(Current).X2 = (LineTranEnd(Current).X2 * 2 + LineTranStart(Current).X1) / 3
                LineTranMid(Current).Y1 = (LineTranEnd(Current).Y2 + LineTranStart(Current).Y1 * 2) / 3
                LineTranMid(Current).Y2 = (LineTranEnd(Current).Y2 * 2 + LineTranStart(Current).Y1) / 3

                LineTranEnd(Current).X1 = LineTranMid(Current).X2
                LineTranEnd(Current).Y1 = LineTranMid(Current).Y2

                LineTranStart(Current).X2 = LineTranMid(Current).X1
                LineTranStart(Current).Y2 = LineTranMid(Current).Y1

                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2, LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2, LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
             End If
          End If
          CurrentControlID = LALLineTransmission(Current)
          AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
       End If
       SaveMe = True 'MODEL IS CHANGED NOW
   End If
End Sub

Private Sub RectLinePoint1_MouseUp(button As Integer, Shift As Integer, X As Single, Y As Single)
   Editing = False
   If button = 2 Then PopupMenu MainWindow.mnuControlInfo
   If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
End Sub
'*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/


'THE FOLLOWING CODE ENCLOSED BY *+*+*+
'IS RESPOSIBLE FOR THE CHANGE OF X1,Y1 POINT OF THE CURRENT LINE.
'IT ALSO DISPLAY A LABEL AS A SQUARE BOX AT X1,Y1 TO GIVE USER RESIZING EFFACT
'*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+
```

```
Private Sub RectLinePoint2_MouseDown(button As Integer, Shift As Integer, X As Single, Y As Single)
   Editing = True
   If button = 2 Then
      If bTransmissionLineBend2.IsExist(Current + 1) Then
         LineTranStart(Current).X2 = (LineTranStart(Current).X1 + LineTranMid(Current).X2) / 2
         LineTranStart(Current).Y2 = (LineTranStart(Current).Y1 + LineTranMid(Current).Y2) / 2
         LineTranMid(Current).X1 = LineTranStart(Current).X2
         LineTranMid(Current).Y1 = LineTranStart(Current).Y2
      Else
         LineTranMid(Current).X1 = (LineTranEnd(Current).X2 + LineTranStart(Current).X1 * 2) / 3
         LineTranMid(Current).X2 = (LineTranEnd(Current).X2 * 2 + LineTranStart(Current).X1) / 3
         LineTranMid(Current).Y1 = (LineTranEnd(Current).Y2 + LineTranStart(Current).Y1 * 2) / 3
         LineTranMid(Current).Y2 = (LineTranEnd(Current).Y2 * 2 + LineTranStart(Current).Y1) / 3
         LineTranEnd(Current).X1 = LineTranMid(Current).X2
         LineTranEnd(Current).Y1 = LineTranMid(Current).Y2
         LineTranStart(Current).X2 = LineTranMid(Current).X1
         LineTranStart(Current).Y2 = LineTranMid(Current).Y1
         RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2, LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
      End If
      RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2, LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
      CurrentControlID = LALLineTransmission(Current)
      AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
      bTransmissionLineBend1.DeleteAtIndex Current + 1
   End If
   If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
End Sub
Private Sub RectLinePoint2_MouseMove(button As Integer, Shift As Integer, X As Single, Y As Single)

   If Editing = True Then
      RectLinePoint2.Move RectLinePoint2.Left + X - RectLinePoint2.Width / 2, RectLinePoint2.Top + Y - RectLinePoint2.Height / 2
      If EditLineBusbar = True Then
         LineBusbar(Current).X2 = RectLinePoint2.Left + RectLinePoint2.Width / 2
         LineBusbar(Current).Y2 = RectLinePoint2.Top + RectLinePoint2.Height / 2
         CurrentControlID = LALLineBusbar(Current)
         AdjustLineLabel LineBusbar(Current), LALLineBusbar(Current)
      ElseIf EditLineTransmission = True Then
         If bTransmissionLineBend2.IsExist(Current + 1) = True Then
            LineTranMid(Current).X1 = RectLinePoint2.Left + RectLinePoint1.Width / 2
            LineTranMid(Current).Y1 = RectLinePoint2.Top + RectLinePoint1.Height / 2
            LineTranStart(Current).X2 = LineTranMid(Current).X1
            LineTranStart(Current).Y2 = LineTranMid(Current).Y1
         Else
               LineTranMid(Current).X1 = RectLinePoint2.Left + RectLinePoint2.Width / 2
               LineTranMid(Current).Y1 = RectLinePoint2.Top + RectLinePoint2.Height / 2
               LineTranStart(Current).X2 = LineTranMid(Current).X1
               LineTranStart(Current).Y2 = LineTranMid(Current).Y1
               LineTranEnd(Current).X1 = (LineTranEnd(Current).X2 + LineTranMid(Current).X1) / 2
```

```
            LineTranEnd(Current).Y1 = (LineTranEnd(Current).Y2 +
LineTranMid(Current).Y1) / 2
            LineTranMid(Current).X2 = LineTranEnd(Current).X1
            LineTranMid(Current).Y2 = LineTranEnd(Current).Y1
            RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
        End If
        CurrentControlID = LALLineTransmission(Current)
        AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
        bTransmissionLineBend1.OpenNewAtIndex Current + 1
      End If
      SaveMe = True 'MODEL IS CHANGED NOW
   End If

End Sub
Private Sub RectLinePoint2_MouseUp(button As Integer, Shift As Integer, X As Single, Y As
Single)
   Editing = False
   If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
End Sub
'*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+

'THE FOLLOWING CODE ENCLOSED BY *+*+*+
'IS RESPOSIBLE FOR THE CHANGE OF X1,Y1 POINT OF THE CURRENT LINE.
'IT ALSO DISPLAY A LABEL AS A SQUARE BOX AT X1,Y1 TO GIVE USER RESIZING EFFACT
'*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+
Private Sub RectLinePoint3_MouseDown(button As Integer, Shift As Integer, X As Single, Y
As Single)
   Editing = True
   If button = 2 Then
      If bTransmissionLineBend1.IsExist(Current + 1) = True Then
         LineTranEnd(Current).X1 = (LineTranEnd(Current).X2 + LineTranMid(Current).X1) /
2
         LineTranEnd(Current).Y1 = (LineTranEnd(Current).Y2 + LineTranMid(Current).Y1) /
2
         LineTranMid(Current).X2 = LineTranEnd(Current).X1
         LineTranMid(Current).Y2 = LineTranEnd(Current).Y1
      Else
         LineTranMid(Current).X1 = (LineTranEnd(Current).X2 + LineTranStart(Current).X1
* 2) / 3
         LineTranMid(Current).X2 = (LineTranEnd(Current).X2 * 2 +
LineTranStart(Current).X1) / 3
         LineTranMid(Current).Y1 = (LineTranEnd(Current).Y2 + LineTranStart(Current).Y1
* 2) / 3
         LineTranMid(Current).Y2 = (LineTranEnd(Current).Y2 * 2 +
LineTranStart(Current).Y1) / 3
         LineTranEnd(Current).X1 = LineTranMid(Current).X2
         LineTranEnd(Current).Y1 = LineTranMid(Current).Y2
         LineTranStart(Current).X2 = LineTranMid(Current).X1
         LineTranStart(Current).Y2 = LineTranMid(Current).Y1
         RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
      End If
      RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
      CurrentControlID = LALLineTransmission(Current)
      AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
```

```
      bTransmissionLineBend2.DeleteAtIndex Current + 1
      OpenMyInfoForm CurrentControlID, DB, ShowNone
   End If
   If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
End Sub
Private Sub RectLinePoint3_MouseMove(button As Integer, Shift As Integer, X As Single, Y
As Single)

   If Editing = True Then
      RectLinePoint3.Move RectLinePoint3.Left + X - RectLinePoint3.Width / 2,
RectLinePoint3.Top + Y - RectLinePoint3.Height / 2
      If EditLineTransmission = True Then
         If bTransmissionLineBend1.IsExist(Current + 1) = True Then
            LineTranMid(Current).X2 = RectLinePoint3.Left + RectLinePoint3.Width / 2
            LineTranMid(Current).Y2 = RectLinePoint3.Top + RectLinePoint3.Height / 2
            LineTranEnd(Current).X1 = LineTranMid(Current).X2
            LineTranEnd(Current).Y1 = LineTranMid(Current).Y2
         Else
               LineTranMid(Current).X2 = RectLinePoint3.Left + RectLinePoint3.Width / 2
               LineTranMid(Current).Y2 = RectLinePoint3.Top + RectLinePoint3.Height / 2
               LineTranEnd(Current).X1 = LineTranMid(Current).X2
               LineTranEnd(Current).Y1 = LineTranMid(Current).Y2
               LineTranStart(Current).X2 = (LineTranStart(Current).X1 +
LineTranMid(Current).X2) / 2
               LineTranStart(Current).Y2 = (LineTranStart(Current).Y1 +
LineTranMid(Current).Y2) / 2
               LineTranMid(Current).X1 = LineTranStart(Current).X2
               LineTranMid(Current).Y1 = LineTranStart(Current).Y2
               RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
               RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
         End If
         CurrentControlID = LALLineTransmission(Current)
         AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
         bTransmissionLineBend2.OpenNewAtIndex Current + 1
      End If
      SaveMe = True 'MODEL IS CHANGED NOW
   End If

End Sub
Private Sub RectLinePoint3_MouseUp(button As Integer, Shift As Integer, X As Single, Y As
Single)
   If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
   Editing = False
End Sub
'*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*

Sub DisableAllSquares()
   'Infact RectLinePoint1 and RectLinePoint2 are two labels used to display "a line is
selected"
   'to unselect a line I make both unvisible
   RectLinePoint1.Visible = False
   RectLinePoint2.Visible = False
   RectLinePoint3.Visible = False
   RectLinePoint4.Visible = False
   'Also change the currentControlId to null because now no any component is selected
   CurrentControlID = ""
```

```
End Sub

Sub UnSelect()
   'shpBOX IS A SHAPE CONTROL USED TO SHOW SOME IMAGE(GENERATOR etc) SELECTED
   shpBOX.Visible = False  'TO MAKE EFFECT OF UNSELECTED
End Sub

Sub ResetAllFlags()

   'THESE FLAGS ARE SET TO FALSE. IT MEAN NO ANT COMPONENT CAN BE EDITED
   EditLineCable = False
   EditLineBusbar = False
   EditLineTransmission = False
   EditTransformer = False
   EditGenerator = False
   EditCuircuitBreaker = False
   EditFeeder = False
   EditSource = False
   Editing = False
   shpBOX.Visible = False

End Sub


Public Sub SaveFile()
   Dim FSys1 As New FileSystemObject
   'THIS PROCEDURE CAN BE CALLED BY THE MAINWINDOW
   'THIS PROCEDURE ONLY BEHAVES AS "MS WORDS" SAVE MENU
   If MyFileName = "" Then
      MainWindow.CommonDialog.Filter = "*.mdb| Drawing Database .mdb" 'ONLY ALLOWING mdb FILES
      MainWindow.CommonDialog.ShowSave    'DISPLAYING SHOWSAVE
      MyFileName = MainWindow.CommonDialog.FileName   'GETTING FILE NAME
      'if MainWindow.Commondialog.CancelError=
      If Trim(MyFileName) = "" Then Exit Sub      'IF FILE NAME IS EMPTY I SHALL NOT DO ANYTHING AND EXIT FROM PROCEDURE

      Caption = MyFileName    'GET FILE NAME

      'UPTO "Loop" I AM REMOVING LEFT PATH OF FILE AND JUST GETTING IT'S NAME
      If InStr(MyFileName, ".mdb") Then Caption = Left(Caption, Len(Caption) - 4)
      Do While InStr(Caption, "\")
         Caption = Right(Caption, Len(Caption) - InStr(Caption, "\"))
      Loop

      DB.Close    'CLOSING OLD FILE
      CreateDB AppPath & Caption & ".bk", AppPath & "\Model" & MyID + 1 & ".bk" 'MAKING A COPY FROM OLD FILE WITH FILENAME.BK IN APPLICATION FOLDER
      CreateDB MyFileName, AppPath & Caption & ".bk" 'MAKING COPY FROM .BK TO THE EXACT LOCATION AND PATH OF NEWFILENAME
      Set DB = OpenDatabase(AppPath & Caption & ".bk", 2, 0)     'OPENING .BK FILE
      FSys1.DeleteFile AppPath & "Model" & MyID + 1 & ".bk"       'DELETING PRIVIOUS BAKUP FILE

   Else    'AREADY SAVED ONCE
      DB.Close
```

```
      CreateDB MyFileName, AppPath & Caption & ".bk"      'ONLY COPY .BK FILE TO THE
EXACT LOACTION AND PATH  OF NEWFILENAME
      Set DB = OpenDatabase(AppPath & Caption & ".bk")
   End If
   SaveMe = False 'HAVE SAVE THE MODEL NOW

End Sub


Sub ImageMouseDown(IMG As Image, X As Single, Y As Single, LAL As Label,
RotationAngle As Integer)
   'THIS FUNCTION IS CALLED ON IMG LEFT CLICK

   If ButtonKey <> "" Then
      PicDrawing_MouseDown 1, 0, IMG.Left + X, IMG.Top + Y
      Exit Sub
   End If
   Editing = True  'MAKING EDITING OF THE IMG IMAGE TRUE

   'MOVING THE MIDDLE OF CONTROL AND IT'S LABEL TO THE MOUSE'S X,Y
COORDINATE
   'IMG.Move IMG.Left + X - IMG.Width / 2, IMG.Top + Y - IMG.Height / 2
   ImageMouseMove IMG, X, Y, LAL, RotationAngle
   IMG.MousePointer = 15 'MAKE MOUSE POINTER A CROSS
   CurrentControlID = LAL  'CURRENTLY SELECTED CONTROL ID

End Sub
Sub ImageMouseMove(IMG As Image, X As Single, Y As Single, LAL As Label, RotationAngle
As Integer)

   If ButtonKey <> "" Then
      PicDrawing_MouseMove 1, 0, IMG.Left + X, IMG.Top + Y
      Exit Sub
   End If

   'IF EDITING IS NOT TURE SHALL NOT MOVE CONTROL.IT MEAN THAT USER MOVE THE
MOUSE ON
   'CONTROL WITHOUT ANY REASON
   If Editing Then
      'MOVING THE MIDDLE OF CONTROL AND IT'S LABEL TO THE MOUSE'S X,Y
COORDINATE
      Dim p As Point
      IMG.Move IMG.Left + X - IMG.Width / 2, IMG.Top + Y - IMG.Height / 2
      If RotationAngle = 90 Then
         LAL.Move IMG.Left + IMG.Width + 60, IMG.Top + IMG.Height / 2 - LAL.Height / 2
      ElseIf RotationAngle = 180 Then
         LAL.Move IMG.Left + IMG.Width / 2 - LAL.Width / 2, IMG.Top - LAL.Height - 60
      ElseIf RotationAngle = 270 Then
         LAL.Move IMG.Left - LAL.Width - 60, IMG.Top + IMG.Height / 2 - LAL.Height / 2
      Else
         LAL.Move IMG.Left + IMG.Width / 2 - LAL.Width / 2, IMG.Top + IMG.Height + 60
      End If
      shpBOX.Visible = False
      SaveMe = True
   End If

End Sub
Sub ImageMouseUp(IMG As Image, X As Single, Y As Single, button As Integer, LAL As
Label)
```

```
   If ButtonKey <> "" Then
      PicDrawing_MouseUp 1, 0, IMG.Left + X, IMG.Top + Y
      Exit Sub
   End If

      IMG.MousePointer = 0 'MAKING POINTER TO DEFAULT

      'SHALL MAKE THE IMAGE IMG TO LOOK LIKE SELECTED  SO THE FOLLWOING 5
LINES
      'MOVE THE SHAP CONTROL AROOUND THE CONTROL AND MAKE IT VISIBLE
      shpBOX.Width = IMG.Width + 120
      shpBOX.Height = IMG.Height + 120
      shpBOX.Left = IMG.Left + IMG.Width / 2 - shpBOX.Width / 2
      shpBOX.Top = IMG.Top + IMG.Height / 2 - shpBOX.Height / 2
      shpBOX.Visible = True

   If Editing = True Then  'HAVE MOVED CONTROL SO I UPDATE DATABASE
      OpenMyInfoForm CurrentControlID, DB, ShowNone    'LASTPARAMETER "ShowNone"
ONLY GET INFORMATION AND SAVE IT IN DATABASE WITHOUT DISPLAYING ANY FORM
      Editing = False 'HAVE EDITED CONTROL NOW
   ElseIf button = 2 Then  'A RIGHT CLICK
      CurrentControlID = LAL
      MainWindow.mnu_ApplyFailureCondition.Visible = False
      MainWindow.mnu_RemoveFailureCondition.Visible = False
      PopupMenu MainWindow.mnuControlInfo 'OPENING A POPUP MENU
   End If

End Sub

Sub SelectLine(X As Single, Y As Single)
'100 % Complete
'THE FOLLOWING PROCEDURE IS CREATED TO SELECT A LINE CONTROL ON THE FORM
'SINCE LINE CONTROL DON'T SUPPORT USER EVENTS. SO ONLY POSSIBLE SOLUTION
'TO CAPTURE THE LINE CONTROL BY THE X,Y COORDINATES OF MOUSE AND
MATHEMATICS

'I HAVE DEVELOPED THIS CODE BY USING BASIC CONCEPT OF 2D GEOMETERY
'THE FOLLOWING CONCEPTS OF 2D GEOMERTRY ARE USED
'   1.CO-ORDINATE PLANE     2.EQUATION OF STRAIGHT LINE    3.SLOPE OF A LINE
'   4.SLOPE OF THE NORMAL   5.MID-POINT                    6.POLAR CO-ORDINATES
'   7.POINT OF A LINE LIE A SPECIFIC DISTANCE AWAY FROM THE LINE EDGE
'   8.PLANE AS PART OF XY-COORDINATE PLANE
'   9.DISTANCE OF A POINT FORM A LINE,
'IF YOU DON'T HAVE THESE CONCEPTS YOU MAY BE UNABLE TO UNDERSTAND THE
FOLLOWIN CODE
'AND ALSO THE CODE OF "AdjustLineLabel" PROCEDURE

'SINCE USER CANNOT EXACTLY CLICK ON THE LINE. SO I HAVE ASSUMED THAT
'IF THERE IS A CLICK LESS THAN 40 PIXELS FAR FROM THE LINE, IS ON THE LINE

   Dim Total As Integer    'COUNTER UPPER LIMIT
   Dim I As Integer    ' COUNTER
   Dim M As Double 'SLOPE OF LINE
   Dim YEst As Double, XEst As Double  'NEAREST POSSIBLE VALUE TO X,Y ON THE LINE

   Total = ldLineBusbar.Total  'GETTING TOTAL BUSBAR LINES
   For I = 0 To Total  'CHECKING ALL BUSBAR LINES
      If ldLineBusbar.IsExist(I + 1) Then 'LINE EXIST(OR LINE IS LOADED)
```

```
        ' CHECKING THE CLICK BOUNDERY OF A BOX(WHICH CAN SURROUND LINE
STRICKTLY)
        If (X < LineBusbar(I).X1 + 40 And X > LineBusbar(I).X2 - 40 Or _
          X > LineBusbar(I).X1 - 40 And X < LineBusbar(I).X2 + 40) And _
          (Y < LineBusbar(I).Y1 + 40 And Y > LineBusbar(I).Y2 - 40 Or _
          Y > LineBusbar(I).Y1 - 40 And Y < LineBusbar(I).Y2 + 40) _
        Then
          'LINE IS APPROXIMATELY PARALLEL TO Y-AXISES
          If Abs(LineBusbar(I).X1 - LineBusbar(I).X2) < 40 Then
            'Can say the following is equal
            If Abs(X - LineBusbar(I).X1) < 40 Then 'CLICKED IS NEAR TO LINE
              Current = I
              RectLinePoint1.Move LineBusbar(Current).X1 - RectLinePoint1.Width / 2,
LineBusbar(Current).Y1 - RectLinePoint1.Height / 2
              RectLinePoint2.Move LineBusbar(Current).X2 - RectLinePoint2.Width / 2,
LineBusbar(Current).Y2 - RectLinePoint2.Height / 2
              RectLinePoint1.Visible = True
              RectLinePoint2.Visible = True
              CurrentControlID = "B" & I + 1
              EditLineBusbar = True
              Exit Sub
            End If

          'LINE IS APPROXIMATELY PARALLEL TO X-AXISES
          ElseIf Abs(LineBusbar(I).Y1 - LineBusbar(I).Y2) < 40 Then 'CLICKED IS NEAR
TO LINE
            If Abs(Y - LineBusbar(I).Y1) < 40 Then
              Current = I
              RectLinePoint1.Move LineBusbar(Current).X1 - RectLinePoint1.Width / 2,
LineBusbar(Current).Y1 - RectLinePoint1.Height / 2
              RectLinePoint2.Move LineBusbar(Current).X2 - RectLinePoint2.Width / 2,
LineBusbar(Current).Y2 - RectLinePoint2.Height / 2
              RectLinePoint1.Visible = True
              RectLinePoint2.Visible = True
              CurrentControlID = "B" & I + 1
              EditLineBusbar = True
              Exit Sub
            End If
          Else 'LINE IS INCLINDED AT SOME ANGLE OTHERE THAN 0,90,180,270
DEGREES

            'SLOPE OF THE LINE
            M = CDbl(LineBusbar(I).Y2 - LineBusbar(I).Y1) / (LineBusbar(I).X2 -
LineBusbar(I).X1)

            YEst = LineBusbar(I).Y1 + M * (X - LineBusbar(I).X1) 'NEAREST POSSIBLE
VALUE OF Y
            XEst = LineBusbar(I).X1 + 1 / M * (Y - LineBusbar(I).Y1) 'NEAREST
POSSIBLE VALUE OF X

            'EITHER THE X IS VERY NEAR TO ESTIMATED X OR EITHER THE Y IS VERY
NEAR TO ESTIMATED Y
            If Abs(YEst - Y) < 40 Or Abs(XEst - X) < 40 Then
              Current = I 'LINE IS SELECTED

              'FOLLOWING 4 SATATEMENTS DISPLAY AS LINE IS SELECTED
              RectLinePoint1.Move LineBusbar(Current).X1 - RectLinePoint1.Width / 2,
LineBusbar(Current).Y1 - RectLinePoint1.Height / 2
```

```
                RectLinePoint2.Move  LineBusbar(Current).X2  -  RectLinePoint2.Width  /  2,
LineBusbar(Current).Y2 - RectLinePoint2.Height / 2
                RectLinePoint1.Visible = True
                RectLinePoint2.Visible = True

                CurrentControlID = "B" & I + 1 'MAKING CURRENT COTROL ID
                EditLineBusbar = True   'NOW EDITING LINE
                Exit Sub    'NO NEED TO CHECK ANY OTHER LINE
            End If
        End If  'MULTIPLE IF CALUSE CLOSING
    End If '(X < LineBusbar(i).X1 + 40 And X > LineBusbar(i).X2 - 40 Or _
            X > LineBusbar(i).X1 - 40 And X < LineBusbar(i).X2 + 40) And _
            (Y < LineBusbar(i).Y1 + 40 And Y > LineBusbar(i).Y2 - 40 Or _
            Y > LineBusbar(i).Y1 - 40 And Y < LineBusbar(i).Y2 + 40)
  End If 'ldLineBusbar.IsExist(i + 1)
Next
'THE FOLLOWING IS SAME FOR THE TRANSMISSION LINES
Total = ldLineTransmission.Total
For I = 0 To Total
  If ldLineTransmission.IsExist(I + 1) Then
    If ( _
      (X < LineTranStart(I).X1 + 40 And X > LineTranStart(I).X2 - 40 Or _
      X > LineTranStart(I).X1 - 40 And X < LineTranStart(I).X2 + 40) And _
      (Y < LineTranStart(I).Y1 + 40 And Y > LineTranStart(I).Y2 - 40 Or _
      Y > LineTranStart(I).Y1 - 40 And Y < LineTranStart(I).Y2 + 40) _
      ) Then
      If (Abs(LineTranStart(I).X1 - LineTranStart(I).X2) < 40) Then
        If Abs(X - LineTranStart(I).X1) < 40 Then
          Current = I
          RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
          RectLinePoint2.Move  LineTranMid(Current).X1  -  RectLinePoint2.Width  /  2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
          RectLinePoint3.Move  LineTranMid(Current).X2  -  RectLinePoint3.Width  /  2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
          RectLinePoint4.Move  LineTranEnd(Current).X2  -  RectLinePoint4.Width  /  2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
          RectLinePoint1.Visible = True
          RectLinePoint2.Visible = True
          RectLinePoint3.Visible = True
          RectLinePoint4.Visible = True
          CurrentControlID = "TL" & I + 1
          EditLineTransmission = True
          Exit Sub
        End If
      ElseIf Abs(LineTranStart(I).Y1 - LineTranStart(I).Y2) < 40 Then
        If Abs(Y - LineTranStart(I).Y1) < 40 Then
          Current = I
          RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
          RectLinePoint2.Move  LineTranMid(Current).X1  -  RectLinePoint2.Width  /  2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
          RectLinePoint3.Move  LineTranMid(Current).X2  -  RectLinePoint3.Width  /  2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
          RectLinePoint4.Move  LineTranEnd(Current).X2  -  RectLinePoint4.Width  /  2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
          RectLinePoint1.Visible = True
          RectLinePoint2.Visible = True
          RectLinePoint3.Visible = True
```

```
                RectLinePoint4.Visible = True
                CurrentControlID = "TL" & I + 1
                EditLineTransmission = True
                Exit Sub

            End If
        Else
            M = CDbl(LineTranStart(I).Y2 - LineTranStart(I).Y1) / (LineTranStart(I).X2 -
LineTranStart(I).X1)
            YEst = LineTranStart(I).Y1 + M * (X - LineTranStart(I).X1)
            XEst = LineTranStart(I).X1 + 1 / M * (Y - LineTranStart(I).Y1)
            If Abs(YEst - Y) < 40 Or Abs(XEst - X) < 40 Then
                Current = I
                RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                RectLinePoint1.Visible = True
                RectLinePoint2.Visible = True
                RectLinePoint3.Visible = True
                RectLinePoint4.Visible = True
                CurrentControlID = "TL" & I + 1
                EditLineTransmission = True
                Exit Sub
            End If
        End If
    End If
  End If
Next

Total = ldLineTransmission.Total
For I = 0 To Total
  If ldLineTransmission.IsExist(I + 1) Then
    If ( _
      (X < LineTranEnd(I).X1 + 40 And X > LineTranEnd(I).X2 - 40 Or _
      X > LineTranEnd(I).X1 - 40 And X < LineTranEnd(I).X2 + 40) And _
      (Y < LineTranEnd(I).Y1 + 40 And Y > LineTranEnd(I).Y2 - 40 Or _
      Y > LineTranEnd(I).Y1 - 40 And Y < LineTranEnd(I).Y2 + 40) _
      ) Then
      If (Abs(LineTranEnd(I).X1 - LineTranEnd(I).X2) < 40) Then
        If Abs(X - LineTranEnd(I).X1) < 40 Then
          Current = I
          RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
          RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
          RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
          RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
          RectLinePoint1.Visible = True
          RectLinePoint2.Visible = True
          RectLinePoint3.Visible = True
          RectLinePoint4.Visible = True
          CurrentControlID = "TL" & I + 1
```

```
                EditLineTransmission = True
                Exit Sub
            End If
        ElseIf Abs(LineTranEnd(I).Y1 - LineTranEnd(I).Y2) < 40 Then
            If Abs(Y - LineTranEnd(I).Y1) < 40 Then
                Current = I
                RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                RectLinePoint1.Visible = True
                RectLinePoint2.Visible = True
                RectLinePoint3.Visible = True
                RectLinePoint4.Visible = True
                CurrentControlID = "TL" & I + 1
                EditLineTransmission = True
                Exit Sub
            End If
        Else
            M = CDbl(LineTranEnd(I).Y2 - LineTranEnd(I).Y1) / (LineTranEnd(I).X2 -
LineTranEnd(I).X1)
            YEst = LineTranEnd(I).Y1 + M * (X - LineTranEnd(I).X1)
            XEst = LineTranEnd(I).X1 + 1 / M * (Y - LineTranEnd(I).Y1)
            If Abs(YEst - Y) < 40 Or Abs(XEst - X) < 40 Then
                Current = I
                RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                RectLinePoint1.Visible = True
                RectLinePoint2.Visible = True
                RectLinePoint3.Visible = True
                RectLinePoint4.Visible = True
                CurrentControlID = "TL" & I + 1
                EditLineTransmission = True
                Exit Sub
            End If
        End If
    End If
End If
Next



Total = ldLineTransmission.Total
For I = 0 To Total
    If ldLineTransmission.IsExist(I + 1) Then
        If ( _
            (X < LineTranMid(I).X1 + 40 And X > LineTranMid(I).X2 - 40 Or _
            X > LineTranMid(I).X1 - 40 And X < LineTranMid(I).X2 + 40) And _
            (Y < LineTranMid(I).Y1 + 40 And Y > LineTranMid(I).Y2 - 40 Or _
```

```
            Y > LineTranMid(I).Y1 - 40 And Y < LineTranMid(I).Y2 + 40) _
            ) Then
            If (Abs(LineTranMid(I).X1 - LineTranMid(I).X2) < 40) Then
               If Abs(X - LineTranMid(I).X1) < 40 Then
                  Current = I
                  RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                  RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                  RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                  RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                  RectLinePoint1.Visible = True
                  RectLinePoint2.Visible = True
                  RectLinePoint3.Visible = True
                  RectLinePoint4.Visible = True
                  CurrentControlID = "TL" & I + 1
                  EditLineTransmission = True
                  Exit Sub
               End If
            ElseIf Abs(LineTranMid(I).Y1 - LineTranMid(I).Y2) < 40 Then
               If Abs(Y - LineTranMid(I).Y1) < 40 Then
                  Current = I
                  RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                  RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                  RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                  RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                  RectLinePoint1.Visible = True
                  RectLinePoint2.Visible = True
                  RectLinePoint3.Visible = True
                  RectLinePoint4.Visible = True
                  CurrentControlID = "TL" & I + 1
                  EditLineTransmission = True
                  Exit Sub
               End If
            Else
               M = CDbl(LineTranMid(I).Y2 - LineTranMid(I).Y1) / (LineTranMid(I).X2 -
LineTranMid(I).X1)
               YEst = LineTranMid(I).Y1 + M * (X - LineTranMid(I).X1)
               XEst = LineTranMid(I).X1 + 1 / M * (Y - LineTranMid(I).Y1)
               If Abs(YEst - Y) < 40 Or Abs(XEst - X) < 40 Then
                  Current = I
                  RectLinePoint1.Move LineTranStart(Current).X1 - RectLinePoint1.Width / 2,
LineTranStart(Current).Y1 - RectLinePoint1.Height / 2
                  RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                  RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                  RectLinePoint4.Move LineTranEnd(Current).X2 - RectLinePoint4.Width / 2,
LineTranEnd(Current).Y2 - RectLinePoint4.Height / 2
                  RectLinePoint1.Visible = True
                  RectLinePoint2.Visible = True
                  RectLinePoint3.Visible = True
                  RectLinePoint4.Visible = True
```

```
                CurrentControlID = "TL" & I + 1
                EditLineTransmission = True
                Exit Sub
            End If
        End If
    End If
  End If
Next
End Sub

Sub AdjustLineLabel(lLINE As Line, LAL As Label)
'THIS FUNCTION USE SOME OF THE CONCEPT OF 2D GEOMETERY, DEFINED IN
"SelectLine" procedure

  If Abs(lLINE.X1 - lLINE.X2) < 20 Then   'Line parallel to Y-Axis
    LAL.Move lLINE.X1 + 80, (lLINE.Y1 + lLINE.Y2) / 2
  ElseIf Abs(lLINE.Y1 - lLINE.Y2) < 20 Then 'Line parallel to X-Axis
    LAL.Move (lLINE.X1 + lLINE.X2) / 2, lLINE.Y1 + 80
  Else        'all other cases
    '   T for TAN,   S for SIN ,  C for COS
    Dim T As Double, S As Double, c As Double
    'T is the slope of the Normal to the line--> Tangent(thita) of normal line
    T = -1 / (CDbl(lLINE.Y2 - lLINE.Y1) / (lLINE.X2 - lLINE.X1))
    c = Sqr(1 / (1 + T * T))    'Cos(thita)
    S = Sqr(1 - c * c)  'Sin(thita)
    'Using polar co-ordinate from mid-point towards normal
    LAL.Left = (lLINE.X1 + lLINE.X2) / 2 + c * 100
    LAL.Top = (lLINE.Y1 + lLINE.Y2) / 2 + S * 100
  End If

End Sub

Public Sub clearlal()

  Dim I As Integer

  For I = 0 To ldLineTransmission.TotalLoaded - 1
    If ldLineTransmission.IsExist(I + 1) Then
      LALPowerFlow(I).Visible = False
      LALlineloss(I).Visible = False
    Else
      LALPowerFlow(ldLineTransmission.TotalLoaded).Visible = False
      LALlineloss(ldLineTransmission.TotalLoaded).Visible = False
    End If
  Next
  For I = 0 To ldGenerator.TotalLoaded - 1
    If ldGenerator.IsExist(I + 1) Then
      LALGen(I).Visible = False
    Else
      LALGen(ldGenerator.TotalLoaded).Visible = False
    End If
  Next
  For I = 0 To ldFeeder.TotalLoaded - 1
    If ldFeeder.IsExist(I + 1) Then
      Lalfee(I).Visible = False
    Else
      Lalfee(ldFeeder.TotalLoaded).Visible = False
    End If
  Next
```

```
    For I = 0 To ldLineBusbar.TotalLoaded - 1
        If ldLineBusbar.IsExist(I + 1) Then
            Lallinebus(I).Visible = False
        Else
            Lallinebus(ldLineBusbar.TotalLoaded).Visible = False
        End If
    Next

End Sub

Public Sub DeleteItem(Optional button As Integer)
    'This function is responsible for the Opening a control
    Dim ToolID As String
    ToolID = CurrentControlID
    If ToolID = "" Then Exit Sub
    Dim Ch1 As String * 1, Ch2 As String * 1, Index As Integer
    Dim Rs As Recordset
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        ldGenerator.DeleteAtIndex Index + 1
        imgGenerator(Index).Visible = False
        LALGenerator(Index).Visible = False
        If Index <> 0 Then Unload imgGenerator(Index)
        If Index <> 0 Then Unload LALGenerator(Index)
        Set Rs = DB.OpenRecordset("generators", 2, 2)
        Rs.FindFirst "GENERATORSID='" & ToolID & "'"
        If Rs.NoMatch = False Then Rs.Delete
        Rs.Close
    ElseIf Ch1 = "F" Then   'Feeder or Load point
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        ldFeeder.DeleteAtIndex Index + 1
        imgFeeder(Index).Visible = False
        LALFeeder(Index).Visible = False
        If Index <> 0 Then Unload imgFeeder(Index)
        If Index <> 0 Then Unload LALFeeder(Index)
        Set Rs = DB.OpenRecordset("FEEDER PARAMETERS", 2, 2)
        Rs.FindFirst ("FEEDERID='" & ToolID) & "'"
        If Rs.NoMatch = False Then Rs.Delete
        Rs.Close
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        ldCircuitBreaker.DeleteAtIndex Index + 1
        LALCircuitbreaker(Index).Visible = 0
        imgCircuitbreaker(Index).Visible = 0
        If Index <> 0 Then Unload imgCircuitbreaker(Index)
        If Index <> 0 Then Unload LALCircuitbreaker(Index)
        Set Rs = DB.OpenRecordset("CIRCUIT BREAKERS", 2, 2)
        Rs.FindFirst "CIRCUITBREAKERSID='" & ToolID & "'"
        If Rs.NoMatch = False Then Rs.Delete
        Rs.Close
    ElseIf Ch1 = "B" Then   'Line Busbar
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            ldLineBusbar.DeleteAtIndex Index + 1
            LALLineBusbar(Index).Visible = 0
            LineBusbar(Index).Visible = 0
            LineBusbar(Index).X1 = 0
            LineBusbar(Index).Y1 = 0
```

```
        LineBusbar(Index).X2 = 0
        LineBusbar(Index).Y2 = 0
        If Index <> 0 Then Unload LineBusbar(Index)
        If Index <> 0 Then Unload LALLineBusbar(Index)
        Set Rs = DB.OpenRecordset("BUSBARS", 2, 2)
        Rs.FindFirst "BUSBARID='" & ToolID & "'"
        If Rs.NoMatch = False Then Rs.Delete
        Rs.Close
    ElseIf Ch1 = "T" Then
       If Ch2 = "L" Then   'Transmission Line
          Index = Right(ToolID, Len(ToolID) - 2) - 1
          ldLineTransmission.DeleteAtIndex Index + 1
          LALLineTransmission(Index).Visible = 0
          LineTranStart(Index).Visible = 0
          LineTranEnd(Index).Visible = 0
          LineTranMid(Index).Visible = 0
          LineTranMid(Index).X1 = 0
          LineTranMid(Index).Y1 = 0
          LineTranMid(Index).X2 = 0
          LineTranMid(Index).Y2 = 0

          LineTranEnd(Index).X1 = 0
          LineTranEnd(Index).Y1 = 0
          LineTranEnd(Index).X2 = 0
          LineTranEnd(Index).Y2 = 0

          LineTranStart(Index).X1 = 0
          LineTranStart(Index).Y1 = 0
          LineTranStart(Index).X2 = 0
          LineTranStart(Index).Y2 = 0

          If Index <> 0 Then Unload LineTranStart(Index)
          If Index <> 0 Then Unload LineTranMid(Index)
          If Index <> 0 Then Unload LineTranEnd(Index)
          If Index <> 0 Then Unload LALLineTransmission(Index)
          Set Rs = DB.OpenRecordset("TRANSMISSION LINES", 2, 2)
          Rs.FindFirst "TRANSMISSIONLINESID='" & ToolID & "'"
          If Rs.NoMatch = False Then Rs.Delete
          Rs.Close
       Else        'Transformer
          Index = Right(ToolID, Len(ToolID) - 1) - 1
          ldtransformer.DeleteAtIndex Index + 1
          LALTransformer(Index).Visible = 0
          imgTransformer(Index).Visible = 0
          If Index <> 0 Then Unload imgTransformer(Index)
          If Index <> 0 Then Unload LALTransformer(Index)
          Set Rs = DB.OpenRecordset("TRANSFORMERS", 2, 2)
          Rs.FindFirst "TRANSFORMERSID='" & ToolID & "'"
          If Rs.NoMatch = False Then Rs.Delete
          Rs.Close
       End If
    End If

    If MainWindow.Toolbar3.Buttons(1).MixedState = True Then
       fMainForm(ActiveFormID).DisplayFlow
    ElseIf MainWindow.Toolbar3.Buttons(8).MixedState = True Then
       fMainForm(ActiveFormID).Combi
    End If
```

```
    ResetAllFlags
    DisableAllSquares
    CurrentControlID = ""
End Sub

Public Sub OpenToolOnForm(ToolID As String, X1 As Integer, Y1 As Integer, RotationAngle As Integer, Optional lalx As Integer, Optional laly As Integer, Optional X2 As Integer, Optional Y2 As Integer, Optional MidX1 As Integer, Optional Midy1 As Integer, Optional MidX2 As Integer, Optional Midy2 As Integer)
    'This function is responsible for the Opening a control at only Opening File Time

    Dim Ch1 As String * 1, Ch2 As String * 1, Index As Integer
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        If Index <> 0 And ldGenerator.IsExist(Index + 1) = False Then Load imgGenerator(Index)
        If Index <> 0 And ldGenerator.IsExist(Index + 1) = False Then Load LALGenerator(Index)
        ldGenerator.OpenNewAtIndex Index + 1
        LALGenerator(Index) = ToolID
        ldGenerator.RotationAngle(Index) = RotationAngle
        ImageOpenMove imgGenerator(Index), X1, Y1, LALGenerator(Index), lalx, laly, RotationAngle
        imgGenerator(Index).Picture = Resource.Generator(RotationAngle / 90 + 1).Picture
        LALGenerator(Index).Visible = True
        imgGenerator(Index).Visible = True
    ElseIf Ch1 = "F" Then   'Feeder or Load point
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        If Index <> 0 And ldFeeder.IsExist(Index + 1) = False Then Load imgFeeder(Index)
        If Index <> 0 And ldFeeder.IsExist(Index + 1) = False Then Load LALFeeder(Index)
        ldFeeder.OpenNewAtIndex Index + 1
        LALFeeder(Index) = ToolID
        ldFeeder.RotationAngle(Index) = RotationAngle
        ImageOpenMove imgFeeder(Index), X1, Y1, LALFeeder(Index), lalx, laly, RotationAngle
        imgFeeder(Index).Picture = Resource.Feeder(RotationAngle / 90 + 1).Picture
        LALFeeder(Index).Visible = True
        imgFeeder(Index).Visible = True
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        If Index <> 0 And ldCircuitBreaker.IsExist(Index + 1) = False Then Load imgCircuitbreaker(Index)
        If Index <> 0 And ldCircuitBreaker.IsExist(Index + 1) = False Then Load LALCircuitbreaker(Index)
        ldCircuitBreaker.OpenNewAtIndex Index + 1
        LALCircuitbreaker(Index) = ToolID
        ldCircuitBreaker.RotationAngle(Index) = RotationAngle
        ImageOpenMove imgCircuitbreaker(Index), X1, Y1, LALCircuitbreaker(Index), lalx, laly, RotationAngle
        imgCircuitbreaker(Index).Picture = Resource.Circuitbreaker(RotationAngle / 90 + 1).Picture
        LALCircuitbreaker(Index).Visible = True
        imgCircuitbreaker(Index).Visible = True

    ElseIf Ch1 = "B" Then   'Line Busbar
          Index = Right(ToolID, Len(ToolID) - 1) - 1
```

```
        If Index <> 0 And ldLineBusbar.IsExist(Index + 1) = False Then Load
LineBusbar(Index)
        If Index <> 0 And ldLineBusbar.IsExist(Index + 1) = False Then Load
LALLineBusbar(Index)
        ldLineBusbar.OpenNewAtIndex Index + 1
        LALLineBusbar(Index) = ToolID
        LineBusbar(Index).X1 = X1
        LineBusbar(Index).X2 = X2
        LineBusbar(Index).Y1 = Y1
        LineBusbar(Index).Y2 = Y2
        AdjustLineLabel LineBusbar(Index), LALLineBusbar(Index)
        LALLineBusbar(Index).Move lalx, laly
        LALLineBusbar(Index).Visible = True
        LineBusbar(Index).Visible = True

   ElseIf Ch1 = "T" Then
      If Ch2 = "L" Then   'Transmission Line
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        If Index <> 0 And ldLineTransmission.IsExist(Index + 1) = False Then Load
LineTranStart(Index)
        If Index <> 0 And ldLineTransmission.IsExist(Index + 1) = False Then Load
LineTranMid(Index)
        If Index <> 0 And ldLineTransmission.IsExist(Index + 1) = False Then Load
LineTranEnd(Index)
        If Index <> 0 And ldLineTransmission.IsExist(Index + 1) = False Then Load
LALLineTransmission(Index)
        ldLineTransmission.OpenNewAtIndex Index + 1
        LALLineTransmission(Index) = ToolID
        LineTranStart(Index).X1 = X1
        LineTranStart(Index).Y1 = Y1
        LineTranStart(Index).X2 = MidX1
        LineTranStart(Index).Y2 = Midy1
        LineTranMid(Index).X1 = MidX1
        LineTranMid(Index).X2 = MidX2
        LineTranMid(Index).Y1 = Midy1
        LineTranMid(Index).Y2 = Midy2
        LineTranEnd(Index).X1 = MidX2
        LineTranEnd(Index).Y1 = Midy2
        LineTranEnd(Index).X2 = X2
        LineTranEnd(Index).Y2 = Y2
        If Abs((X1 + MidX2) / 2 - MidX1) > 20 And Abs((Y1 + Midy2) / 2 - Midy1) > 20
Then bTransmissionLineBend1.OpenNewAtIndex Index + 1
        If Abs((X1 + MidX1) / 2 - MidX2) > 20 And Abs((Y1 + Midy1) / 2 - Midy2) > 20
Then bTransmissionLineBend2.OpenNewAtIndex Index + 1
        AdjustLineLabel LineTranMid(Index), LALLineTransmission(Index)
        LALLineTransmission(Index).Move lalx, laly
        LALLineTransmission(Index).Visible = True
        LineTranStart(Index).Visible = True
        LineTranMid(Index).Visible = True
        LineTranEnd(Index).Visible = True
      Else      'Transformer
        Index = Right(ToolID, Len(ToolID) - 1) - 1

        If Index <> 0 And ldtransformer.IsExist(Index + 1) = False Then Load
imgTransformer(Index)
        If Index <> 0 And ldtransformer.IsExist(Index + 1) = False Then Load
LALTransformer(Index)
        ldtransformer.OpenNewAtIndex Index + 1
        LALTransformer(Index) = ToolID
```

```
            ldtransformer.RotationAngle(Index) = RotationAngle
            ImageOpenMove imgTransformer(Index), X1, Y1, LALTransformer(Index), lalx, laly,
RotationAngle
            imgTransformer(Index).Picture = Resource.Transformer(RotationAngle / 90 +
1).Picture
            LALTransformer(Index).Visible = True
            imgTransformer(Index).Visible = True

        End If
    End If
    DoEvents
End Sub

Sub ImageOpenMove(IMG As Image, X As Integer, Y As Integer, LAL As Label, X1 As
Integer, Y1 As Integer, RotationAngle As Integer)
    'Moving a image and it's label to x,y
    IMG.Move X, Y
    '"If RotationAngle = 90 Then
        '"LAL.Move IMG.Left + IMG.Width + 60, IMG.Top + IMG.Height / 2 - LAL.Height / 2
    LAL.Move X1, Y1
    LAL.Visible = True
    '"ElseIf RotationAngle = 180 Then
        '"LAL.Move IMG.Left + IMG.Width / 2 - LAL.Width / 2, IMG.Top - LAL.Height - 60
    '"ElseIf RotationAngle = 270 Then
        '"LAL.Move IMG.Left - LAL.Width - 60, IMG.Top + IMG.Height / 2 - LAL.Height / 2
    '"Else
        '"LAL.Move IMG.Left + IMG.Width / 2 - LAL.Width / 2, IMG.Top + IMG.Height + 60
    '"End If

End Sub

Public Sub StartBlinking(ToolID As String)
    'This function is to add somecontrol to blink
    'Infact it adds the information of the control's label
    'to an array. A timer running to get if and change it;s forcolor
    TimerBlinker = True
    If ToolID = "" Then Exit Sub
    Dim Ch1 As String * 1, Ch2 As String * 1, Index As Integer
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        AddToBlinkArray LALGenerator(Index)
    ElseIf Ch1 = "F" Then   'Feeder or Load point
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        AddToBlinkArray LALFeeder(Index)
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        AddToBlinkArray LALCircuitbreaker(Index)
    ElseIf Ch1 = "B" Then   'Line Busbar
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            AddToBlinkArray LALLineBusbar(Index)
    ElseIf Ch1 = "T" Then
        If Ch2 = "L" Then   'Transmission Line
            Index = Right(ToolID, Len(ToolID) - 2) - 1
            AddToBlinkArray LALLineTransmission(Index)
        Else        'Transformer
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            AddToBlinkArray LALTransformer(Index)
```

```
        End If
    End If

End Sub

Public Sub StopBlinking(ToolID As String)
    'This function is responsible for the Opening a control
    If ToolID = "" Then Exit Sub
    Dim Ch1 As String * 1, Ch2 As String * 1, Index As Integer
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        RemoveFromBlink LALGenerator(Index)
    ElseIf Ch1 = "F" Then   'Feeder or Load point
        Index = Right(ToolID, Len(ToolID) - 1) - 1
        RemoveFromBlink LALFeeder(Index)
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        RemoveFromBlink LALCircuitbreaker(Index)
    ElseIf Ch1 = "B" Then   'Line Busbar
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            RemoveFromBlink LALLineBusbar(Index)
    ElseIf Ch1 = "T" Then
        If Ch2 = "L" Then   'Transmission Line
            Index = Right(ToolID, Len(ToolID) - 2) - 1
            RemoveFromBlink LALLineTransmission(Index)
        Else        'Transformer
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            RemoveFromBlink LALTransformer(Index)
        End If
    End If

End Sub


Public Sub AddToBlinkArray(LAL As String)
    'This is a supporting function of Sub "StartBlinking"
    Dim I As Integer
    I = ldBlinkLAL.NewItemNumber
    If I > UBound(BlinkLAL) Then ReDim Preserve BlinkLAL(UBound(BlinkLAL) + 1)
        Dim iIndex As Integer
        iIndex = 0
        If IsNumeric(Right(LAL, Len(LAL) - 1)) Then
            iIndex = Right(LAL, Len(LAL) - 1)
        Else
            iIndex = Right(LAL, Len(LAL) - 2)
        End If
        iIndex = iIndex - 1
        If InStr(LAL, "TL") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALLineTransmission(iIndex)
        ElseIf InStr(LAL, "T") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALTransformer(iIndex)
        ElseIf InStr(LAL, "F") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALFeeder(iIndex)
        ElseIf InStr(LAL, "CB") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALCircuitbreaker(iIndex)
        ElseIf InStr(LAL, "B") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALLineBusbar(iIndex)
```

```
        ElseIf InStr(LAL, "G") = 1 Then
            Set BlinkLAL(UBound(BlinkLAL)) = LALGenerator(iIndex)
        End If
    TimerBlinker.Enabled = True

End Sub

Sub RemoveFromBlink(LAL As Label)
    'This is supporting function of Sub "StopBlinking "
    ldBlinkLAL.DeleteAtIndex LAL.Index + 1

End Sub


'THE FOLLOWING CODE ENCLOSED BY */*/*/
'IS RESPOSIBLE FOR THE CHANGE OF X1,Y1 POINT OF THE CURRENT LINE.
'IT ALSO DISPLAY A LABEL AS A SQUARE BOX AT X1,Y1 TO GIVE USER RESIZING EFFACT
'*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
/*/*/*/*/*/*/*/
Private Sub RectLinePoint4_MouseDown(button As Integer, Shift As Integer, X As Single, Y
As Single)
    Editing = True
End Sub
Private Sub RectLinePoint4_MouseMove(button As Integer, Shift As Integer, X As Single, Y
As Single)
    If Editing = True Then
        RectLinePoint4.Move RectLinePoint4.Left + X - RectLinePoint4.Width / 2,
RectLinePoint4.Top + Y - RectLinePoint4.Height / 2
        If EditLineTransmission = True Then
            If bTransmissionLineBend2.IsExist(Current + 1) = True Then
                LineTranEnd(Current).X2 = RectLinePoint4.Left + RectLinePoint4.Width / 2
                LineTranEnd(Current).Y2 = RectLinePoint4.Top + RectLinePoint4.Height / 2
            Else
                If bTransmissionLineBend1.IsExist(Current + 1) = True Then
                    LineTranEnd(Current).X2 = RectLinePoint4.Left + RectLinePoint4.Width / 2
                    LineTranEnd(Current).Y2 = RectLinePoint4.Top + RectLinePoint4.Height / 2
                    LineTranEnd(Current).X1 = (LineTranEnd(Current).X2 +
LineTranMid(Current).X1) / 2
                    LineTranEnd(Current).Y1 = (LineTranEnd(Current).Y2 +
LineTranMid(Current).Y1) / 2
                    LineTranMid(Current).X2 = (LineTranEnd(Current).X2 +
LineTranMid(Current).X1) / 2
                    LineTranMid(Current).Y2 = (LineTranEnd(Current).Y2 +
LineTranMid(Current).Y1) / 2
                    RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                    RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
                Else
                    LineTranEnd(Current).X2 = RectLinePoint4.Left + RectLinePoint4.Width / 2
                    LineTranEnd(Current).Y2 = RectLinePoint4.Top + RectLinePoint4.Height / 2
                    LineTranMid(Current).X1 = (LineTranEnd(Current).X2 +
LineTranStart(Current).X1 * 2) / 3
                    LineTranMid(Current).X2 = (LineTranEnd(Current).X2 * 2 +
LineTranStart(Current).X1) / 3
                    LineTranMid(Current).Y1 = (LineTranEnd(Current).Y2 +
LineTranStart(Current).Y1 * 2) / 3
                    LineTranMid(Current).Y2 = (LineTranEnd(Current).Y2 * 2 +
LineTranStart(Current).Y1) / 3
```

```
                LineTranEnd(Current).X1 = LineTranMid(Current).X2
                LineTranEnd(Current).Y1 = LineTranMid(Current).Y2

                LineTranStart(Current).X2 = LineTranMid(Current).X1
                LineTranStart(Current).Y2 = LineTranMid(Current).Y1

                RectLinePoint2.Move LineTranMid(Current).X1 - RectLinePoint2.Width / 2,
LineTranMid(Current).Y1 - RectLinePoint2.Height / 2
                RectLinePoint3.Move LineTranMid(Current).X2 - RectLinePoint3.Width / 2,
LineTranMid(Current).Y2 - RectLinePoint3.Height / 2
            End If
        End If
        CurrentControlID = LALLineTransmission(Current)
        AdjustLineLabel LineTranMid(Current), LALLineTransmission(Current)
    End If
    SaveMe = True 'MODEL IS CHANGED NOW
  End If
End Sub

Private Sub RectLinePoint4_MouseUp(button As Integer, Shift As Integer, X As Single, Y As
Single)
  Editing = False
  If button = 2 Then PopupMenu MainWindow.mnuControlInfo
  If button = 1 Then OpenMyInfoForm CurrentControlID, DB, ShowNone
End Sub
'*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
/*/*/*/*/*/*/*/

Private Sub TimerBlinker_Timer()
  'This timer is infact blinks the controls of which reference is found in BlinkLaL array
  'TimerBlinker = False
  Dim I As Integer
  On Error Resume Next
  For I = 0 To UBound(BlinkLAL)
    If ldBlinkLAL.IsExist(I + 1) Then
      If BlinkLAL(I).ForeColor = vbRed Then
        BlinkLAL(I).FontBold = True
        BlinkLAL(I).ForeColor = vbBlue
      Else
        BlinkLAL(I).FontBold = False
        BlinkLAL(I).ForeColor = vbRed
      End If
    End If
  Next
End Sub


Private Function ShowFlow(ByVal InitialSegment As Integer, FlowLine As Line, Optional
PositiveDirection As Boolean = True, Optional Color1 As Long = 13816530, Optional Color2
As Long = 255) As Integer
  'Initial segment must lie between 0 and 500, and
  '0-250   means White color first
  '250-500 means black color first
  'PicDrawing.Line (FlowLine.X1, FlowLine.Y1)-(FlowLine.X2, FlowLine.Y2), RGB(255, 255,
255)
  On Error Resume Next
  Static BWFlag As Boolean 'It is flag to indicate Black Line or White Line
  If InitialSegment >= 250 Then
```

```
      If InitialSegment = 500 Then
         InitialSegment = 250
      Else
         InitialSegment = InitialSegment Mod 250
      End If
      BWFlag = True
   Else
      BWFlag = False
   End If
   Static I As Integer, PreX As Double, PreY As Double, _
         curX As Double, curY As Double   'Cur stands for current
   Static Distance As Double
   Distance = Sqr((FlowLine.X1 - FlowLine.X2) ^ 2 + (FlowLine.Y1 - FlowLine.Y2) ^ 2)

   If PositiveDirection Then
      PreX = FlowLine.X1
      PreY = FlowLine.Y1
      Do While (InitialSegment < Distance)
         curX = (FlowLine.X2 * InitialSegment + FlowLine.X1 * (Distance - InitialSegment))
/ Distance
         curY = (FlowLine.Y2 * InitialSegment + FlowLine.Y1 * (Distance - InitialSegment))
/ Distance
         If BWFlag Then
            'Black Lines
            PicDrawing.Line (PreX, PreY)-(curX, curY), Color2
            BWFlag = False
         Else
            'White Lines
            PicDrawing.Line (PreX, PreY)-(curX, curY), Color1
            BWFlag = True
         End If
         InitialSegment = InitialSegment + 250
         PreX = curX
         PreY = curY
      Loop
         If BWFlag Then
            'Black Line
            PicDrawing.Line (PreX, PreY)-(FlowLine.X2, FlowLine.Y2), Color2
            ShowFlow = InitialSegment - Distance + 250
         Else
            'White Line
            PicDrawing.Line (PreX, PreY)-(FlowLine.X2, FlowLine.Y2), Color1
            ShowFlow = InitialSegment - Distance ' + 250
         End If

   Else 'Reverse Direction
      PreX = FlowLine.X2
      PreY = FlowLine.Y2
      Do While (InitialSegment < Distance)
         curX = (FlowLine.X1 * InitialSegment + FlowLine.X2 * (Distance - InitialSegment))
/ Distance
         curY = (FlowLine.Y1 * InitialSegment + FlowLine.Y2 * (Distance - InitialSegment))
/ Distance
         If BWFlag Then
            'Black Lines
            PicDrawing.Line (PreX, PreY)-(curX, curY), Color2
            BWFlag = False
         Else
            'White Lines
```

```
            PicDrawing.Line (PreX, PreY)-(curX, curY), Color1
            BWFlag = True
         End If
         InitialSegment = InitialSegment + 250
         PreX = curX
         PreY = curY
      Loop
         If BWFlag Then
            'Black Line
            PicDrawing.Line (PreX, PreY)-(FlowLine.X1, FlowLine.Y1), Color2
            ShowFlow = InitialSegment - Distance + 250
         Else
            'White Line
            PicDrawing.Line (PreX, PreY)-(FlowLine.X1, FlowLine.Y1), Color1
            ShowFlow = InitialSegment - Distance ' + 250
         End If
   End If

End Function

Private Sub TimerSimulationFlow_Timer()
   Static FlowIncrement As Integer
   Static Temp As Integer
   On Error GoTo StopTimer
   Dim I As Integer
   If WindowState = 1 Then Exit Sub
   FlowIncrement = FlowIncrement + 50
   If FlowIncrement > 500 Then FlowIncrement = 0
   For I = 0 To ldLineTransmission.Total
      Temp = FlowIncrement
      If ldLineTransmission.IsExist(I + 1) Then
         If LineTranStart(I).BorderColor <> vbRed Then
            If AnimationTLS(I) = 1 Then
               Temp = ShowFlow(Temp, LineTranStart(I), True, RGB(160, 220, 160),
RGB(0, 180, 0))
               Temp = ShowFlow(Temp, LineTranMid(I), True, RGB(160, 220, 160), RGB(0,
180, 0))
               Temp = ShowFlow(Temp, LineTranEnd(I), True, RGB(160, 220, 160), RGB(0,
180, 0))
               '"Temp = ShowFlow(Temp, LineTranStart(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
               '"Temp = ShowFlow(Temp, LineTranMid(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
               '"Temp = ShowFlow(Temp, LineTranEnd(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
            ElseIf AnimationTLS(I) = 2 Then
               Temp = ShowFlow(Temp, LineTranStart(I), True, RGB(220, 160, 160),
RGB(180, 0, 0))
               Temp = ShowFlow(Temp, LineTranMid(I), True, RGB(220, 160, 160),
RGB(180, 0, 0))
               Temp = ShowFlow(Temp, LineTranEnd(I), True, RGB(220, 160, 160),
RGB(180, 0, 0))
               '"Temp = ShowFlow(Temp, LineTranStart(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
               '"Temp = ShowFlow(Temp, LineTranMid(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
               '"Temp = ShowFlow(Temp, LineTranEnd(I), True, RGB(180, 220, 180),
RGB(0, 175, 0))
            ElseIf AnimationTLS(I) = -1 Then
```

```
                Temp = ShowFlow(Temp, LineTranStart(I), False, RGB(160, 220, 220),
RGB(0, 180, 180))
                Temp = ShowFlow(Temp, LineTranMid(I), False, RGB(160, 220, 220), RGB(0,
180, 180))
                Temp = ShowFlow(Temp, LineTranEnd(I), False, RGB(160, 220, 220), RGB(0,
180, 180))
                '"Temp = ShowFlow(Temp, LineTranStart(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))
                '"Temp = ShowFlow(Temp, LineTranMid(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))
                '"Temp = ShowFlow(Temp, LineTranEnd(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))
            ElseIf AnimationTLS(I) = -2 Then
                Temp = ShowFlow(Temp, LineTranStart(I), False, RGB(220, 160, 220),
RGB(180, 0, 0))
                Temp = ShowFlow(Temp, LineTranMid(I), False, RGB(220, 160, 220),
RGB(180, 0, 0))
                Temp = ShowFlow(Temp, LineTranEnd(I), False, RGB(220, 160, 220),
RGB(180, 0, 0))
                '"Temp = ShowFlow(Temp, LineTranStart(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))
                '"Temp = ShowFlow(Temp, LineTranMid(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))
                '"Temp = ShowFlow(Temp, LineTranEnd(I), False, RGB(180, 220, 180),
RGB(0, 175, 0))

            End If
          End If
        End If
    Next
'       If FlowIncrement = 0 Or FlowIncrement = 150 Or FlowIncrement = 300 Or
FlowIncrement = 450 Then
'       For I = 0 To ldLineBusbar.Total
'          Temp = FlowIncrement
'          Temp = ShowFlow(255 * Rnd, LineBusbar(I), True, RGB(220, 220, 220), RGB(0, 0,
0))
'       Next
'   End If
    Exit Sub
StopTimer:
    TimerSimulationFlow.Enabled = False
End Sub


Public Function N_to_N_Detection(Optional Prompt As Boolean = True) As Boolean
        'Node to Node Detection and Save information into Database
'THIS PROCEDURE WILL DO THE FOLLOWINGS
'   1.  IT DETECT ALL THE EDGES OF THE ALL TRANSMISSION LINES
'   2.  IT STORE THE INFORMATION OF TRANSMISSION LINE CONNECTED TO AND FROM
'       IN THE DATABASE
'    3.  AFTER DETECTION OF THE EDGES OF THE TRANSMISSION LINES, IT SHALL
PERFORMS
'         SOME NECESSARY OPERATIONS ON THE DATABASE, AND ADJUST THE OTHER
COMPONENTS'
'           CONNECTED TO AND FROM. IT ALSO INDICATES THE COMPONENTS NOT
CONNECTED TO THE CIRCUIT
    On Error GoTo ShowError
    Set rsForAll = DB.OpenRecordset("TRANSMISSION LINES", dbOpenDynaset, False)
    ResetAllFlags
```

```
    '1
    Dim I As Integer, T As Integer, J As Integer
    Dim I2 As Integer, T2 As Integer
    Dim X As Single, Y As Single, p As Point
    Dim CrtlID As String

    N_to_N_Detection = False
    frmGraphTable.Clear     'clearing Graph table

    '############################   CHECKING TRANSMISSION LINES WITH
ALL OTHER CONTROLS ####################################
    T = ldLineTransmission.Total
    For I = 0 To T  'CHECK ALL THE LINES
        If ldLineTransmission.IsExist(I + 1) Then 'IT MEAN LINE IS DRAWN ON THE FORM
            rsForAll.FindFirst "TransmissionLinesID='" & LALLineTransmission(I) & "'"
            rsForAll.Edit
            If rsForAll.NoMatch = True Then
                MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
                End
            End If

            For J = 1 To 2
                If J = 1 Then
                    X = LineTranStart(I).X1
                    Y = LineTranStart(I).Y1
                Else
                    X = LineTranEnd(I).X2
                    Y = LineTranEnd(I).Y2
                End If
                'CHECKING GENERATORS
                For I2 = 0 To ldGenerator.Total
                    If ldGenerator.IsExist(I2 + 1) Then 'GENERATOR EXIST
                        If  N_to_N_Helper(imgGenerator(I2),  X,  Y,  ldGenerator.RotationAngle(I2))
Then
                            If NoControlConnected(CrtlID, I, I2) Then Exit Function
                            If J = 1 Then
                                rsForAll.Fields("Connected From") = LALGenerator(I2)
                                frmGraphTable.Add LALLineTransmission(I)
                                frmGraphTable.AddConnectedFrom LALGenerator(I2)
                            ElseIf J = 2 Then
                                rsForAll.Fields("Connected To") = LALGenerator(I2)
                                frmGraphTable.AddConnectedTo LALGenerator(I2)
                            End If
                            CrtlID = LALGenerator(I2)
                        End If
                    End If
                Next

                'CHECKING TRANSFORMERS
                For I2 = 0 To ldtransformer.Total
                    If ldtransformer.IsExist(I2 + 1) Then 'tRANSFORMER EXIST
                        If              N_to_N_Helper(imgTransformer(I2),              X,              Y,
ldtransformer.RotationAngle(I2)) Then
                            If NoControlConnected(CrtlID, I, I2) Then Exit Function
                            If J = 1 Then
                                rsForAll.Fields("Connected From") = LALTransformer(I2)
                                frmGraphTable.Add LALLineTransmission(I)
                                frmGraphTable.AddConnectedFrom LALTransformer(I2)
                            ElseIf J = 2 Then
```

```
                        rsForAll.Fields("Connected To") = LALTransformer(I2)
                        frmGraphTable.AddConnectedTo LALTransformer(I2)
                    End If
                        CrtlID = LALTransformer(I2)
                End If
            End If
        Next
        'CHECKING FEEDERS
        For I2 = 0 To ldFeeder.Total
            If ldFeeder.IsExist(I2 + 1) Then 'FEEDER EXIST
                If N_to_N_Helper(imgFeeder(I2), X, Y, ldFeeder.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALFeeder(I2)
                        frmGraphTable.Add LALLineTransmission(I)
                        frmGraphTable.AddConnectedFrom LALFeeder(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALFeeder(I2)
                        frmGraphTable.AddConnectedTo LALFeeder(I2)
                    End If
                    CrtlID = LALFeeder(I2)
                End If
            End If
        Next

        'CHECKING CIRCUITBREAKERS
        For I2 = 0 To ldCircuitBreaker.Total
            If ldCircuitBreaker.IsExist(I2 + 1) Then 'CIRCUITBREAKER EXIST
                If N_to_N_Helper(imgCircuitbreaker(I2), X, Y,
ldCircuitBreaker.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALCircuitbreaker(I2)
                        frmGraphTable.Add LALLineTransmission(I)
                        frmGraphTable.AddConnectedFrom LALCircuitbreaker(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALCircuitbreaker(I2)
                        frmGraphTable.AddConnectedTo LALCircuitbreaker(I2)
                    End If
                    CrtlID = LALCircuitbreaker(I2)
                End If
            End If
        Next
        'Checking Busbar
        SelectLine X, Y
        If CurrentControlID <> "" Then
            If Left(CurrentControlID, 1) = "B" Then
                'HENCE THIS LINE CONNECTED TO BUSBAR
                 If CrtlID <> "" Then
                    If Prompt = True Then MsgBox "Confliction detected among the
Transmission Line(" & LALLineTransmission(I) & ")," & vbNewLine & "and componets with
IDs " & CrtlID & " AND " & CurrentControlID & vbNewLine & _
                        "Node to node detection teriminated. Please adjust the conflicted
components", vbInformation
                    DisableAllSquares
                    Exit Function
                End If
                If J = 1 Then
                    rsForAll.Fields("Connected From") = CurrentControlID
```

```
                    frmGraphTable.Add LALLineTransmission(I)
                    frmGraphTable.AddConnectedFrom CurrentControlID
                ElseIf J = 2 Then
                    rsForAll.Fields("Connected To") = CurrentControlID
                    frmGraphTable.AddConnectedTo CurrentControlID
                End If
                CrtlID = CurrentControlID
                DisableAllSquares
            End If
        End If

        If CrtlID = "" Then
            If Prompt = True Then MsgBox "One or both ends of the Transmission Line(" & LALLineTransmission(I) & ")" & _
                vbNewLine & "is not connected to any component" & vbNewLine & _
                "Node to node detection teriminated. Please adjust it", vbInformation
            DisableAllSquares
            Exit Function
        Else
            CrtlID = ""
        End If

    Next
        rsForAll.Update
    End If  ' End Transmission Line Exist
  Next
  '############################## END->CHECKING TRANSMISSION LINES WITH ALL OTHER CONTROLS ####################################
  rsForAll.Close

  '##############################    CHECKING FEEDER WITH ALL OTHER CONTROLS ####################################
  'ONLY CHECK FEEDERS WITH BUSBARS
  Set rsForAll = DB.OpenRecordset("Feeder Parameters", 2, 0)
  For I = 0 To ldFeeder.Total 'CHECKING ALL FEEDERS
    If ldFeeder.IsExist(I + 1) Then
      rsForAll.FindFirst "FeederID='" & LALFeeder(I) & "'"
      If rsForAll.NoMatch = True Then
        MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
        End
      End If
      p = RotationPoint(imgFeeder(I), imgFeeder(I).Left + 16 * 15, imgFeeder(I).Top, ldFeeder.RotationAngle(I))
      SelectLine CSng(p.X), CSng(p.Y)
      If CurrentControlID <> "" Then
        If Left(CurrentControlID, 1) = "B" Then
          'HENCE THIS FEEDER CONNECTED TO BUSBAR
          rsForAll.Edit
          rsForAll.Fields("Connected From") = CurrentControlID
          rsForAll.Update
          frmGraphTable.Add LALFeeder(I)
          frmGraphTable.AddConnectedFrom CurrentControlID
          CrtlID = CurrentControlID
          DisableAllSquares
        End If
      Else
        If Prompt = True Then MsgBox "Feeder(" & LALFeeder(I) & ") is not connected to any busbar" & vbNewLine & _
            "Node to node detection teriminated. Please adjust it", vbInformation
```

```
            DisableAllSquares
            Exit Function
         End If
      End If
   Next
   rsForAll.Close
   '############################ END-> CHECKING FEEDER WITH ALL
OTHER CONTROLS ####################################

   '############################  CHECKING TRANSFORMERS WITH ALL
OTHER CONTROLS ####################################
   Set rsForAll = DB.OpenRecordset("TRANSFORMERS", 2, 0)
   T = ldtransformer.Total
   For I = 0 To T  'CHECK ALL THE LINES
      If ldtransformer.IsExist(I + 1) Then 'IT MEAN LINE IS DRAWN ON THE FORM
         rsForAll.FindFirst "TRANSFORMERSID='" & LALTransformer(I) & "'"
         rsForAll.Edit
         If rsForAll.NoMatch = True Then
            MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
            End
         End If
         CrtlID = ""
         For J = 1 To 2
            If J = 1 Then
               p = RotationPoint(imgTransformer(I), imgTransformer(I).Left - 15,
imgTransformer(I).Top + 17 * 15, ldtransformer.RotationAngle(I))
               X = p.X
               Y = p.Y
            Else
               p = RotationPoint(imgTransformer(I), imgTransformer(I).Left +
imgTransformer(I).Width + 15, imgTransformer(I).Top + 17 * 15,
ldtransformer.RotationAngle(I))
               X = p.X
               Y = p.Y
            End If

            'CHECKING GENERATORS
            For I2 = 0 To ldGenerator.Total
               If ldGenerator.IsExist(I2 + 1) Then 'GENERATOR EXIST
                  If N_to_N_Helper(imgGenerator(I2), X, Y, ldGenerator.RotationAngle(I2))
Then
                     If NoControlConnected(CrtlID, I, I2) Then Exit Function
                     If J = 1 Then
                        rsForAll.Fields("Connected From") = LALGenerator(I2)
                        frmGraphTable.Add LALTransformer(I)
                        frmGraphTable.AddConnectedFrom LALGenerator(I2)
                     ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALGenerator(I2)
                        frmGraphTable.AddConnectedTo LALGenerator(I2)
                     End If
                     CrtlID = LALGenerator(I2)
                  End If
               End If
            Next

            'CHECKING TRANSFORMERS
            For I2 = 0 To ldtransformer.Total
               If ldtransformer.IsExist(I2 + 1) And I <> I2 Then  'tRANSFORMER EXIST
```

```
                If N_to_N_Helper(imgTransformer(I2), X, Y,
ldtransformer.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALTransformer(I2)
                        frmGraphTable.Add LALTransformer(I)
                        frmGraphTable.AddConnectedFrom LALTransformer(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALTransformer(I2)
                        frmGraphTable.AddConnectedTo LALTransformer(I2)
                    End If
                        CrtlID = LALTransformer(I2)
                End If
            End If
        Next
        'CHECKING FEEDERS
        For I2 = 0 To ldFeeder.Total
            If ldFeeder.IsExist(I2 + 1) Then 'FEEDER EXIST
                If N_to_N_Helper(imgFeeder(I2), X, Y, ldFeeder.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALFeeder(I2)
                        frmGraphTable.Add LALTransformer(I)
                        frmGraphTable.AddConnectedFrom LALFeeder(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALFeeder(I2)
                        frmGraphTable.AddConnectedTo LALFeeder(I2)
                    End If
                    CrtlID = LALFeeder(I2)
                End If
            End If
        Next

        'CHECKING CIRCUITBREAKERS
        For I2 = 0 To ldCircuitBreaker.Total
            If ldCircuitBreaker.IsExist(I2 + 1) Then 'CIRCUITBREAKER EXIST
                If N_to_N_Helper(imgCircuitbreaker(I2), X, Y,
ldCircuitBreaker.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALCircuitbreaker(I2)
                        frmGraphTable.Add LALTransformer(I)
                        frmGraphTable.AddConnectedFrom LALCircuitbreaker(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALCircuitbreaker(I2)
                        frmGraphTable.AddConnectedTo LALCircuitbreaker(I2)
                    End If
                    CrtlID = LALCircuitbreaker(I2)
                End If
            End If
        Next
        'Checking Busbar
        SelectLine X, Y
        If CurrentControlID <> "" Then
            'HENCE THIS LINE CONNECTED TO BUSBAR
            If J = 1 Then
                rsForAll.Fields("Connected From") = CurrentControlID
                frmGraphTable.Add LALTransformer(I)
                frmGraphTable.AddConnectedFrom CurrentControlID
```

```
                ElseIf J = 2 Then
                    rsForAll.Fields("Connected To") = CurrentControlID
                    frmGraphTable.AddConnectedTo CurrentControlID
                End If
                CrtlID = CurrentControlID
                DisableAllSquares
            End If

            If CrtlID = "" Then
                If Prompt = True Then MsgBox "One or both ends of the Transformer (" &
LALTransformer(I) & ")" & _
                    vbNewLine & "is not connected to any component" & vbNewLine & _
                    "Node to node detection teriminated. Please adjust it", vbInformation
                DisableAllSquares
                Exit Function
            Else
                CrtlID = ""
            End If

        Next
            rsForAll.Update
        End If  ' End Transmission Line Exist
    Next
    rsForAll.Close
    '############################  END CHECKING TRANSFORMER WITH
ALL OTHER CONTROLS ####################################


    '############################    CHECKING CIRCUIT BREAKERS WITH
ALL OTHER CONTROLS ####################################
    Set rsForAll = DB.OpenRecordset("Circuit BREakerS", 2, 0)
    T = ldCircuitBreaker.Total
    For I = 0 To T  'CHECK ALL THE LINES
        If ldCircuitBreaker.IsExist(I + 1) Then 'IT MEAN LINE IS DRAWN ON THE FORM
            rsForAll.FindFirst "CircuitBREakerSID='" & LALCircuitbreaker(I) & "'"
            rsForAll.Edit
            If rsForAll.NoMatch = True Then
                MsgBox "UNKNOWN ERROR", vbCritical, "ERROR"
                End
            End If
            CrtlID = ""
            For J = 1 To 2
                If J = 1 Then
                    p = RotationPoint(imgCircuitbreaker(I), _
                    imgCircuitbreaker(I).Left - 15, _
                    imgCircuitbreaker(I).Top + 17 * 15, ldCircuitBreaker.RotationAngle(I))
                    X = p.X
                    Y = p.Y
                Else
                    p = RotationPoint(imgCircuitbreaker(I), _
                    imgCircuitbreaker(I).Left + imgCircuitbreaker(I).Width + 15, _
                    imgCircuitbreaker(I).Top + 17 * 15, ldCircuitBreaker.RotationAngle(I))
                    X = p.X
                    Y = p.Y
                End If

                'CHECKING GENERATORS
                For I2 = 0 To ldGenerator.Total
                    If ldGenerator.IsExist(I2 + 1) Then 'GENERATOR EXIST
```

```
                If N_to_N_Helper(imgGenerator(I2), X, Y, ldGenerator.RotationAngle(I2))
Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALGenerator(I2)
                        frmGraphTable.Add LALCircuitbreaker(I)
                        frmGraphTable.AddConnectedFrom LALGenerator(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALGenerator(I2)
                        frmGraphTable.AddConnectedTo LALGenerator(I2)
                    End If
                    CrtlID = LALGenerator(I2)
                End If
            End If
        Next

        'CHECKING CircuitBREakerS
        For I2 = 0 To ldCircuitBreaker.Total
            If ldCircuitBreaker.IsExist(I2 + 1) And I <> I2 Then 'CircuitBREaker EXIST
                If N_to_N_Helper(imgCircuitbreaker(I2), X, Y,
ldCircuitBreaker.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALCircuitbreaker(I2)
                        frmGraphTable.Add LALCircuitbreaker(I)
                        frmGraphTable.AddConnectedFrom LALCircuitbreaker(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALCircuitbreaker(I2)
                        frmGraphTable.AddConnectedTo LALCircuitbreaker(I2)
                    End If
                        CrtlID = LALCircuitbreaker(I2)
                End If
            End If
        Next
        'CHECKING FEEDERS
        For I2 = 0 To ldFeeder.Total
            If ldFeeder.IsExist(I2 + 1) Then 'FEEDER EXIST
                If N_to_N_Helper(imgFeeder(I2), X, Y, ldFeeder.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
                        rsForAll.Fields("Connected From") = LALFeeder(I2)
                        frmGraphTable.Add LALCircuitbreaker(I)
                        frmGraphTable.AddConnectedFrom LALFeeder(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALFeeder(I2)
                        frmGraphTable.AddConnectedTo LALFeeder(I2)
                    End If
                    CrtlID = LALFeeder(I2)
                End If
            End If
        Next

        'CHECKING TransFORmerS
        For I2 = 0 To ldtransformer.Total
            If ldtransformer.IsExist(I2 + 1) Then 'TransFORmer EXIST
                If N_to_N_Helper(imgTransformer(I2), X, Y,
ldtransformer.RotationAngle(I2)) Then
                    If NoControlConnected(CrtlID, I, I2) Then Exit Function
                    If J = 1 Then
```

```
                        rsForAll.Fields("Connected From") = LALTransformer(I2)
                        frmGraphTable.Add LALCircuitbreaker(I)
                        frmGraphTable.AddConnectedFrom LALTransformer(I2)
                    ElseIf J = 2 Then
                        rsForAll.Fields("Connected To") = LALTransformer(I2)
                        frmGraphTable.AddConnectedTo LALTransformer(I2)
                    End If
                    CrtlID = LALTransformer(I2)
                End If
            End If
        Next
        'Checking Busbar
        SelectLine X, Y
        If CurrentControlID <> "" Then
            'HENCE THIS LINE CONNECTED TO BUSBAR
            If J = 1 Then
                rsForAll.Fields("Connected From") = CurrentControlID
                frmGraphTable.Add LALCircuitbreaker(I)
                frmGraphTable.AddConnectedFrom CurrentControlID
            ElseIf J = 2 Then
                rsForAll.Fields("Connected To") = CurrentControlID
                frmGraphTable.AddConnectedTo CurrentControlID
            End If
            CrtlID = CurrentControlID
            DisableAllSquares
        End If

        If CrtlID = "" Then
            If Prompt = True Then MsgBox "One or both ends of the Circuit Breaker(" &
LALCircuitbreaker(I) & ")" & _
                vbNewLine & "is not connected to any component" & vbNewLine & _
                "Node to node detection teriminated. Please adjust it", vbInformation
            DisableAllSquares
            Exit Function
        Else
            CrtlID = ""
        End If

    Next
        rsForAll.Update
    End If  ' End Transmission Line Exist
Next
rsForAll.Close
'############################ END CHECKING CircuitBREaker WITH ALL
OTHER CONTROLS ####################################
'The Following Two are dependent on above four

'Checking Generators

For I = 0 To ldGenerator.Total
    Dim Temp As String
    If ldGenerator.IsExist(I + 1) Then
        Temp = ""
        For J = 1 To frmGraphTable.grdGraph.Rows - 1
            frmGraphTable.grdGraph.Row = J
            frmGraphTable.grdGraph.col = 2
            If frmGraphTable.grdGraph.Text = LALGenerator(I) Then
                frmGraphTable.grdGraph.col = 1
                Temp = frmGraphTable.grdGraph.Text
```

```
                frmGraphTable.Add LALGenerator(I)
                frmGraphTable.AddConnectedTo Temp
                End If
            frmGraphTable.grdGraph.Row = J
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = LALGenerator(I) Then
                frmGraphTable.grdGraph.col = 1
                Temp = frmGraphTable.grdGraph.Text
                frmGraphTable.Add LALGenerator(I)
                frmGraphTable.AddConnectedTo Temp
            End If
        Next
        If Temp = "" Then
            For J = 0 To ldLineBusbar.Total
                If ldLineBusbar.IsExist(J + 1) Then
                    p = GeneratorConnectedToBusbar(imgGenerator(I), LineBusbar(J))
                    If p.X <> -1 Then
                        If N_to_N_Helper(imgGenerator(I), CSng(p.X), CSng(p.Y),
ldGenerator.RotationAngle(I)) Then
                        CurrentControlID = ""
                        If N_to_N_Helper(imgGenerator(I), CSng(LineBusbar(J).X1),
CSng(LineBusbar(J).Y1), ldGenerator.RotationAngle(I)) Or N_to_N_Helper(imgGenerator(I),
CSng(LineBusbar(J).X2), CSng(LineBusbar(J).Y2), ldGenerator.RotationAngle(I)) = True
Then
                            CurrentControlID = LALLineBusbar(J)
                        Else
                            SelectLine CSng(p.X), CSng(p.Y)
                        End If
                        If CurrentControlID = LALLineBusbar(J) Then
                            frmGraphTable.grdGraph.col = 1
                            frmGraphTable.Add LALGenerator(I)
                            frmGraphTable.AddConnectedTo LALLineBusbar(J)
                            Temp = "none"
                        End If
                    End If
                  End If
                End If
            Next
            If Temp = "" Then
                If Prompt = True Then MsgBox "Generator(" & LALGenerator(I) & ")" & _
                    vbNewLine & "is not connected to any component" & vbNewLine & _
                    "Node to node detection teriminated. Please adjust it", vbInformation
                DisableAllSquares
                Exit Function
            End If
        End If
    End If
Next

For I = 0 To ldLineBusbar.Total
    If ldLineBusbar.IsExist(I + 1) Then
        For J = 1 To frmGraphTable.grdGraph.Rows - 1
            frmGraphTable.grdGraph.Row = J
            frmGraphTable.grdGraph.col = 2
            If frmGraphTable.grdGraph.Text = LALLineBusbar(I) Then
                frmGraphTable.grdGraph.col = 1
                Temp = frmGraphTable.grdGraph.Text
                frmGraphTable.Add LALLineBusbar(I)
                frmGraphTable.AddConnectedTo Temp
```

```
            End If
            frmGraphTable.grdGraph.Row = J
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = LALLineBusbar(I) Then
               frmGraphTable.grdGraph.col = 1
               Temp = frmGraphTable.grdGraph.Text
               frmGraphTable.Add LALLineBusbar(I)
               frmGraphTable.AddConnectedTo Temp
            End If
         Next
         If Temp = "" Then
         SelectLine LineBusbar(I).X1, LineBusbar(I).Y1
         If Prompt = True Then MsgBox "Busbar(" & LALLineBusbar(I) & ")" & _
                     vbNewLine & "is not connected to any component" & vbNewLine & _
                     "Node to node detection teriminated. Please adjust it", vbInformation
            DisableAllSquares
            CurrentControlID = ""
            Exit Function
         End If
      End If
   Next
   N_to_N_Detection = True
   DisableAllSquares
   If Prompt = True Then
      Dim response
      response = MsgBox("No Error found in Node to Node detection" & vbNewLine & "Do
you want to view Graph Table", vbYesNo + vbInformation)
      If response = vbYes Then
         frmGraphTable.Show 1
      End If
   End If
   Exit Function
ShowError:
   MsgBox Err.Number & " " & Err.Description
End Function

Private Function N_to_N_Helper(IMG As Image, X As Single, Y As Single, RotationAngle As
Integer) As Boolean
   'This function returns True if a control can be connected at X,Y
   'This control can be "Generator", "Transformer", "Feeder" and "Circuit Breaker" only
   N_to_N_Helper = False
   Dim TopLeftX As Single, TopLeftY As Single
   Dim DownRightX As Single, DownRightY As Single
   TopLeftX = IMG.Left
   TopLeftY = IMG.Top
   DownRightX = TopLeftX + IMG.Width
   DownRightY = TopLeftY + IMG.Height
   If (X > TopLeftX - 15 And X < DownRightX + 15) And _
     (Y > TopLeftY - 15 And Y < DownRightY + 15) Then
      'This cahnge the point(x,y) by rotating it on the image Clock wise
      '-------------------------------------------------------------------------------------------
      'Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)
Rotate X and Y (*)
      X = X - TopLeftX
      Y = Y - TopLeftY
      Dim Temp As Single
      If RotationAngle = 90 Then
         Temp = Y
         Y = X
```

```
            X = IMG.Height - Temp
        ElseIf RotationAngle = 180 Then
            Y = IMG.Height - Y
            X = IMG.Width - X
        ElseIf RotationAngle = 270 Then
            Temp = X
            X = Y
            Y = IMG.Width - Temp
        End If
        X = X + TopLeftX
        Y = Y + TopLeftY
        'Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)
Rotate X and Y (*)
        '-------------------------------------------------------------------------------------------------
        Select Case (IMG.Name)
        Case Is = "imgGenerator" 'Generator OK
            If (X >= TopLeftX + 1.5 * 15 And X <= DownRightX - 2.5 * 15) And _
              (Y >= TopLeftY + 3 * 15 And Y <= DownRightY - 6 * 15) Then
                  N_to_N_Helper = True
                  If X <= TopLeftX + 5 * 15 Then
                     If Y <= TopLeftY + 6 * 15 Or Y >= DownRightY - 9 * 15 Then
N_to_N_Helper = False
                  ElseIf X >= DownRightX - 6 * 15 Then
                     If Y <= TopLeftY + 6 * 15 Or Y >= DownRightY - 9 * 15 Then
N_to_N_Helper = False
                  End If
            ElseIf (X >= TopLeftX + 15 * 15 And X <= DownRightX - 15 * 14) And _
                 (Y >= TopLeftY + 15 * 25 And Y <= DownRightY + 15 * 2) Then
                 N_to_N_Helper = True
            End If
        Case Is = "imgCircuitbreaker" 'Circuit Braker OK
            If Y > TopLeftY + 14 * 15 And Y < DownRightY - 12 * 15 Then N_to_N_Helper =
True
        Case Is = "imgFeeder"
            If Y < TopLeftY + 5 * 15 Then
               If X > TopLeftX + 14 * 15 And X < DownRightX - 14 * 15 Then N_to_N_Helper
= True
            End If
        Case Is = "imgTransformer"
            If Y >= TopLeftY + 14 * 15 And Y <= DownRightY - 12 * 15 Then
               If X < TopLeftX + 5 * 15 Or X > DownRightX - 5 * 15 Then N_to_N_Helper =
True
            End If
        End Select
    End If

End Function

Private Function NoControlConnected(CrtlID As String, I As Integer, I2 As Integer) As
Boolean
    If CrtlID <> "" Then
        MsgBox "Confliction detected among controls. Atleast three controls connected at a
single node" & vbNewLine & _
               "Node to node detection teriminated. Please adjust the conflicted components"
        DisableAllSquares
        NoControlConnected = True
    Else
        NoControlConnected = False
    End If
```

```
End Function

Public Sub Cut()
   If CurrentControlID <> "" Then
      CutCopyPaste.SaveDataFrom DB, CurrentControlID
      DeleteItem
   End If
End Sub

Public Sub Copy()
   If CurrentControlID <> "" Then CutCopyPaste.SaveDataFrom DB, CurrentControlID
End Sub

Public Sub Paste()
   If CutCopyPaste.SaveDataToDB(DB) = True Then
      OpenToolOnForm CutCopyPaste.ToolID, -PicDrawing.Left + 360, -PicDrawing.Top +
360, CutCopyPaste.Rotation, 1100 - PicDrawing.Left + 360, 1100 - PicDrawing.Top + 360,
700 - PicDrawing.Left + 360, 700 - PicDrawing.Top + 360, 900 - PicDrawing.Left + 360,
900 - PicDrawing.Top + 360
   Else
      'ButtonKey = CutCopyPaste.ToolID
      'PicDrawing_MouseDown 1, 0, 0, 0
      'PicDrawing_MouseMove 1, 0, 500, 500
      'PicDrawing_MouseUp 1, 0, 500, 500
      'OpenToolOnForm CutCopyPaste.ToolID, 500, 500, 1100, 1100, 700, 700, 900, 900
   End If
   UnSelect

End Sub

Public Sub ApplyFailureCondition()

   Dim Index As Integer
   Index = Right(CurrentControlID, Len(CurrentControlID) - 2)
   LineTranStart(Index - 1).BorderColor = vbRed
   LineTranMid(Index - 1).BorderColor = vbRed
   LineTranEnd(Index - 1).BorderColor = vbRed
   DisableAllSquares

End Sub

Public Sub RemoveFailureCondition()

   Dim Index As Integer
   Index = Right(CurrentControlID, Len(CurrentControlID) - 2)
   LineTranStart(Index - 1).BorderColor = &HC000&
   LineTranMid(Index - 1).BorderColor = &HC000&
   LineTranEnd(Index - 1).BorderColor = &HC000&
   DisableAllSquares

End Sub

Public Sub ResizeMe()
   Dim K As Long, I As Long, J As Long
   On Error Resume Next
   'Ploting dots on the form
   For I = 0 To PicDrawing.Width Step 144
      For J = 0 To PicDrawing.Height Step 144
         If I Mod 1440 = 0 Or J Mod 1440 = 0 Then
```

```
            K = K + 1
            Load Shape1(CLng(K))
            Shape1(CLng(K)).Left = I
            Shape1(CLng(K)).Top = J
            Shape1(CLng(K)).BorderColor = RGB(255, 0, 0)
            'Shape1(K).Width = 25
            'Shape1(K).Height = 25
            Shape1(CLng(K)).Visible = True
         ElseIf I Mod 1440 / 2 = 0 Or J Mod 1440 / 2 = 0 Then
            K = K + 1
            Load Shape1(CLng(K))
            Shape1(CLng(K)).Left = I
            Shape1(CLng(K)).Top = J
            'Shape1(K).BorderColor = RGB(255, 0, 0)
            'Shape1(K).Width = 25
            'Shape1(K).Height = 25
            Shape1(CLng(K)).Visible = True
         End If
         'DoEvents 'NO NEED HERE, BECAUSE SOME ERRORS OCCURS WHEN FORM
UNLOADED
      Next
   Next
   Shape1(0).Visible = False
   Call Form_Resize
End Sub

Public Sub Rotate()

   'This function is responsible for the Opening a control
   Dim ToolID As String
   ToolID = CurrentControlID
   If ToolID = "" Then Exit Sub
   Dim Ch1 As String * 1, Ch2 As String * 1, Index As Integer
   Dim Rs As Recordset
   Ch1 = ToolID
   Ch2 = Right(ToolID, Len(ToolID) - 1)
   If Ch1 = "G" Then   'Generator
      Index = Right(ToolID, Len(ToolID) - 1) - 1
      Set Rs = DB.OpenRecordset("generators", 2, 2)
      Rs.FindFirst "GENERATORSID='" & ToolID & "'"
      If Rs.NoMatch = False Then
         ldGenerator.AddRotationAngle(Index) = 90
         imgGenerator(Index).Picture                                    =
Resource.Generator(ldGenerator.RotationAngle(Index) / 90 + 1).Picture
         ImageMouseDown imgGenerator(Index), imgGenerator(Index).Width / 2,
imgGenerator(Index).Height / 2, LALGenerator(Index), ldGenerator.RotationAngle(Index)
         ImageMouseUp imgGenerator(Index), imgGenerator(Index).Width / 2,
imgGenerator(Index).Height / 2, 0, LALGenerator(Index)
         Rs.Edit
         Rs.Fields("Rotation") = ldGenerator.RotationAngle(Index)
         Rs.Update
      Else
         MsgBox "Unexpected Error", vbCritical
      End If
      Rs.Close
   ElseIf Ch1 = "F" Then   'Feeder or Load point
      Index = Right(ToolID, Len(ToolID) - 1) - 1
      Set Rs = DB.OpenRecordset("FEEDER PARAMETERS", 2, 2)
      Rs.FindFirst ("FEEDERID='" & ToolID) & "'"
```

```
        If Rs.NoMatch = False Then
            ldFeeder.AddRotationAngle(Index) = 90
            imgFeeder(Index).Picture = Resource.Feeder(ldFeeder.RotationAngle(Index) / 90 +
1).Picture
            ImageMouseDown imgFeeder(Index), imgFeeder(Index).Width / 2,
imgFeeder(Index).Height / 2, LALFeeder(Index), ldFeeder.RotationAngle(Index)
            ImageMouseUp imgFeeder(Index), imgFeeder(Index).Width / 2,
imgFeeder(Index).Height / 2, 0, LALFeeder(Index)
            Rs.Edit
            Rs.Fields("Rotation") = ldFeeder.RotationAngle(Index)
            Rs.Update
        Else
            MsgBox "Unexpected Error", vbCritical
        End If
        Rs.Close
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Index = Right(ToolID, Len(ToolID) - 2) - 1
        Set Rs = DB.OpenRecordset("CIRCUIT BREAKERS", 2, 2)
        Rs.FindFirst "CIRCUITBREAKERSID='" & ToolID & "'"
        If Rs.NoMatch = False Then
            ldCircuitBreaker.AddRotationAngle(Index) = 90
            imgCircuitbreaker(Index).Picture =
Resource.Circuitbreaker(ldCircuitBreaker.RotationAngle(Index) / 90 + 1).Picture
            ImageMouseDown imgCircuitbreaker(Index), imgCircuitbreaker(Index).Width / 2,
imgCircuitbreaker(Index).Height / 2, LALCircuitbreaker(Index),
ldCircuitBreaker.RotationAngle(Index)
            ImageMouseUp imgCircuitbreaker(Index), imgCircuitbreaker(Index).Width / 2,
imgCircuitbreaker(Index).Height / 2, 0, LALCircuitbreaker(Index)
            Rs.Edit
            Rs.Fields("Rotation") = ldCircuitBreaker.RotationAngle(Index)
            Rs.Update
        Else
            MsgBox "Unexpected Error", vbCritical
        End If
        Rs.Close
    ElseIf Ch1 = "T" Then
        If Ch2 = "L" Then   'Transmission Line

        Else       'Transformer
            Index = Right(ToolID, Len(ToolID) - 1) - 1
            Set Rs = DB.OpenRecordset("TRANSFORMERS", 2, 2)
            Rs.FindFirst "TRANSFORMERSID='" & ToolID & "'"
            If Rs.NoMatch = False Then
                ldtransformer.AddRotationAngle(Index) = 90
                imgTransformer(Index).Picture =
Resource.Transformer(ldtransformer.RotationAngle(Index) / 90 + 1).Picture
                ImageMouseDown imgTransformer(Index), imgTransformer(Index).Width / 2,
imgTransformer(Index).Height / 2, LALTransformer(Index),
ldtransformer.RotationAngle(Index)
                ImageMouseUp imgTransformer(Index), imgTransformer(Index).Width / 2,
imgTransformer(Index).Height / 2, 0, LALTransformer(Index)
                Rs.Edit
                Rs.Fields("Rotation") = ldtransformer.RotationAngle(Index)
                Rs.Update
            Else
                MsgBox "Unexpected Error", vbCritical
            End If
            Rs.Close
        End If
```

```
    End If

End Sub

Private Function RotationPoint(IMG As Image, ByVal X As Single, ByVal Y As Single,
RotationAngle As Integer) As Point
    'This function returns the value of the point after rotating it on the image Anti-Clock wise
    '-----------------------------------------------------------------------------------------------
    'Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X
and Y (*)
    X = X - IMG.Left
    Y = Y - IMG.Top
    Dim Temp As Single
    If RotationAngle = 90 Then
        Temp = X
        X = Y
        Y = IMG.Width - Temp
    ElseIf RotationAngle = 180 Then
        Y = IMG.Height - Y
        X = IMG.Width - X
    ElseIf RotationAngle = 270 Then
        Temp = Y
        Y = X
        X = IMG.Height - Temp
    End If
    X = X + IMG.Left
    Y = Y + IMG.Top
    RotationPoint.X = X
    RotationPoint.Y = Y

    'Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X and Y (*)  Rotate X
and Y (*)
    '-----------------------------------------------------------------------------------------------

End Function
```

B.1.17 FrmMaintenanceSchedule

```
Option Explicit
Dim strDate() As New LocalDate
Dim strData() As String

Public Sub Maintain(DB As Database, Flag As Integer)


    Dim I As Integer
    ReDim strData(1, 0)

'#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*
    'THE FOLLOWING SWITCH ONLY ADD ALL DATES OF MANAGMENT
    Select                                  Case                                    Flag
'#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*
        Case Is = 0 'Today
            ReDim strDate(0)
            Set strDate(0) = New LocalDate
```

```
        strDate(0).Value = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") &
"/" & Format(Year(Date), "00")
    Case Is = 1 'Current Week
        ReDim strDate(6)
        Dim DayWeek As Integer, PDate As Date
        DayWeek = Weekday(Date)
        PDate = Date - DayWeek + 1
        Set strDate(0) = New LocalDate
        strDate(0).Value = Format(Day(PDate), "00") & "/" & Format(Month(PDate), "00")
& "/" & Format(Year(PDate), "00")
        For I = 1 To 6
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    Case Is = 2 'Current Month
        ReDim strDate(0)
        strDate(0).Value = 1 & "/" & Month(Date) & "/" & Year(Date)
        If strDate(0).MyMonth = 2 And strDate(0).MyYear Mod 4 <> 0 Then
            ReDim strDate(27)
        ElseIf strDate(0).MyMonth = 2 And strDate(0).MyYear Mod 4 = 0 Then
            ReDim strDate(28)
        ElseIf (strDate(0).MyMonth = 4 Or strDate(0).MyMonth = 6 Or strDate(0).MyMonth
= 9 Or strDate(0).MyMonth = 11) And I = 30 Then
            ReDim strDate(29)
        Else
            ReDim strDate(30)
        End If
        strDate(0).Value = 1 & "/" & Month(Date) & "/" & Year(Date)
        For I = 1 To UBound(strDate)
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    Case Is = 6 'Current Year
        ReDim strDate(365)
        strDate(0).Value = "1/1/" & Format(Year(Date), "00")
        For I = 1 To 365
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    Case Is = 3 'Next 6 Days
        ReDim strDate(6)
        Set strDate(0) = New LocalDate
        strDate(0).Value = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") &
"/" & Format(Year(Date), "00")
        For I = 1 To 6
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    Case Is = 4 'Next 30 days
        ReDim strDate(30)
        strDate(0).Value = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") &
"/" & Format(Year(Date), "00")
        For I = 1 To 30
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    Case Is = 5 'Next 365 days
        ReDim strDate(365)
```

```
        strDate(0).Value = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") &
"/" & Format(Year(Date), "00")
        For I = 1 To 365
            Set strDate(I) = New LocalDate
            strDate(I).Value = strDate(I - 1).AddDays(1)
        Next
    End                                    Select                                    'Flag
'*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*

    frmWait.lalCounter = 0
    frmWait.lalTotal = 7 + UBound(strDate)
    frmWait.Show
    frmWait.Refresh


    Dim CtrlID As String, MDate As New LocalDate, MInterval As Integer
    On Error GoTo WriteError
    Dim Rs As Recordset
    '////////////////////////////////GENERATOR/////////////////////////////////////
    GetInfo DB, "generators", "GeneratorsID", "MaintenanceInterval",
"NextMaintenanceDate"

    '////////////////////////////////FEEDER/////////////////////////////////////
    'GetInfo DB, "Feeder Parameters", "FeederID", "MaintenanceInterval",
"NextMaintenanceDate"

    '////////////////////////////////CIRCUIT BREAKER/////////////////////////////////////
    GetInfo DB, "Circuit Breakers", "CircuitbreakersID", "MaintenanceInterval",
"NextMaintenanceDate"

    '////////////////////////////////TRANSFORMER/////////////////////////////////////
    GetInfo DB, "Transformers", "TransformersID", "MaintenanceInterval",
"NextMaintenanceDate"

    '////////////////////////////////BUSBAR/////////////////////////////////////
    GetInfo DB, "Busbars", "BusbarID", "MaintenanceInterval", "NextMaintenanceDate"

    '////////////////////////////////TRANSMISSION LINE/////////////////////////////////////
    GetInfo DB, "Transmission Lines", "TransmissionLinesID", "MaintenanceInterval",
"NextMaintenanceDate"

    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    SetGrid
    Exit Sub
WriteError:
    If Err.Number = 94 Then
        Resume Next
    Else
        MsgBox Err.Number & "-->" & Err.Description
        Resume Next
    End If
End Sub

Private Sub SetGrid()
    Dim I As Long, J As Long, K As Long
    Dim CurrentRow(1 To 6)
    grdMainReport.Cols = 7
    grdMainReport.Rows = 1
```

```
   grdMainReport.TextMatrix(0, 0) = "Date"
   grdMainReport.TextMatrix(0, 1) = "Generator"
   grdMainReport.TextMatrix(0, 2) = "Feeder"
   grdMainReport.TextMatrix(0, 3) = "Circuit Breaker"
   grdMainReport.TextMatrix(0, 4) = "Busbar"
   grdMainReport.TextMatrix(0, 5) = "Transmission Line"
   grdMainReport.TextMatrix(0, 6) = "Transformer"
   grdMainReport.ColWidth(0) = 1100
   grdMainReport.ColWidth(3) = 1200
   grdMainReport.ColWidth(5) = 1500

   For I = 0 To UBound(strDate)
         For K = 1 To 6
            CurrentRow(K) = grdMainReport.RowHeight(0)
         Next
         grdMainReport.Rows = grdMainReport.Rows + 1
         grdMainReport.Row = grdMainReport.Rows - 1
         grdMainReport.col = 0
         grdMainReport.Text = strDate(I).Value
      For J = 0 To UBound(strData, 2)
         If strData(0, J) = strDate(I).Value Then
            Dim Ch1 As String * 1, Ch2 As String * 1
            Ch1 = strData(1, J)
            Ch2 = Right(strData(1, J), Len(strData(1, J)) - 1)
            If Ch1 = "G" Then   'Generator
               K = 1
            ElseIf Ch1 = "F" Then   'Feeder or Load point
               K = 2
            ElseIf Ch1 = "C" Then   'Cricuit Breaker
               K = 3
            ElseIf Ch1 = "B" Then   'Line Busbar
               K = 4
            ElseIf Ch1 = "T" Then
               If Ch2 = "L" Then   'Transmission Line
                  K = 5
               Else       'Transformer
                  K = 6
               End If
            End If
            grdMainReport.col = K
            If grdMainReport.Text = "" Then
               grdMainReport.Text = strData(1, J)
            Else
               CurrentRow(K) = CurrentRow(K) + grdMainReport.RowHeight(0)
               If grdMainReport.RowHeight(grdMainReport.Rows - 1) < CurrentRow(K) Then
grdMainReport.RowHeight(grdMainReport.Rows - 1) = CurrentRow(K)
               'grdMainReport.Row = CurrentRow(K)
               grdMainReport.Text = grdMainReport.Text & vbNewLine & strData(1, J)
            End If
         End If
      Next
   frmWait.lalCounter = frmWait.lalCounter + 1
   frmWait.lalCounter.Refresh
   Next
   grdMainReport.Height = 5775
   If grdMainReport.Rows * grdMainReport.RowHeight(0) + 220 < grdMainReport.Height
Then
      grdMainReport.Height = grdMainReport.Rows * grdMainReport.RowHeight(0) + 220
      If grdMainReport.Height < 1200 Then
```

```
            grdMainReport.Height = 1200
        End If
    End If
    cmdOk.Top = grdMainReport.Height + 220
    cmdPrint.Top = cmdOk.Top
    Height = grdMainReport.Height + cmdOk.Height + 720

End Sub

Private Sub AddComponent(MDate As LocalDate, strID As String)
    ' Here MDate is the Managing Date and strId is the Id of the control to manage

    Dim I As Long
    If strID = "" Then Exit Sub
    ReDim Preserve strData(1, UBound(strData, 2) + 1)
    'For I = 0 To UBound(strDate)
    '    If strDate(I) = MDate.Value Then
            strData(0, UBound(strData, 2)) = MDate.Value
            strData(1, UBound(strData, 2)) = strID
    '        Exit Sub
    '    End If
    'Next
    Exit Sub ' We do not need to extend the strdate because we alreay extended it

    ReDim Preserve strDate(UBound(strDate) + 1)
    strDate(UBound(strDate)) = Format(Day(MDate), "00") & "/" & Format(Month(MDate),
"00") & "/" & Format(Year(MDate), "00")
    strData(0, UBound(strData, 2)) = strDate(UBound(strDate))
    strData(1, UBound(strData, 2)) = strID


End Sub

Private Function CheckDate(dDate As LocalDate) As Boolean
    CheckDate = False
    If dDate.CompareLocalDate(strDate(0)) >= 0 And
dDate.CompareLocalDate(strDate(UBound(strDate))) <= 0 Then CheckDate = True

End Function

Private Function ChangeToDate(strDate As String) As Date

    Dim D As Integer, M As Integer, Y As Integer
    D = Left(Trim(strDate), InStr(strDate, "/") - 1)
    M = Right(Left(Trim(strDate), 5), 2)
    Y = Right(strDate, Len(strDate) - 6)
    ChangeToDate = M & "/" & D & "/" & Y


End Function

Private Sub cmdOK_Click()
    Hide
End Sub

Private Sub cmdPrint_Click()

    Printer.FontSize = 14
    Printer.Print "Maintinance Schedule of " & Caption
```

```
    Printer.FontSize = 8
    PrintGrid grdMainReport, 500
    Printer.EndDoc

End Sub

Private Sub GetInfo(DB As Database, CtrlTable As String, Ctrl_ID As String, CtrlMInterval As
String, CtrlNextMainDate As String, Optional UpdateData As Boolean = False)
    Dim Rs As Recordset, I As Integer, MInterval As Long, MDate As New LocalDate, CtrlID
As String
    Set Rs = DB.OpenRecordset(CtrlTable) 'OPENING AND DRAWING GENERATOR
    frmWait.lalCounter = frmWait.lalCounter + 1
    frmWait.lalCounter.Refresh
    For I = 0 To Rs.RecordCount - 1 'INITIALIZING LOOP FOR ALL GENERATORS
        CtrlID = ""
        MInterval = 0
        CtrlID = Rs.Fields(Ctrl_ID)
        MInterval = Rs.Fields(CtrlMInterval)
        MDate.Value = Rs.Fields(CtrlNextMainDate)
        If MDate.CompareDate(Date) < 0 And UpdateData = True Then
            Dim Temp As New LocalDate
            Temp.Value = MDate.Value
            Do While Temp.CompareDate(Date) < 0
                Temp.Value = Temp.AddDays(MInterval)
            Loop
            Rs.Edit
            Rs.Fields(CtrlNextMainDate) = Temp.Value
            Rs.Update
            MDate.Value = Temp.Value
        End If

        If CheckDate(MDate) Then
            Do While CheckDate(MDate)
                MDate.Value = MDate.AddDays(-MInterval)
            Loop
            MDate.Value = MDate.AddDays(MInterval)
        End If
        Do While CheckDate(MDate)
            AddComponent MDate, CtrlID
            MDate.Value = MDate.AddDays(MInterval)
        Loop
        Rs.MoveNext
    Next 'END CTRL
    Rs.Close

End Sub
```

B.1.18 FrmPerformanceIndecCal

```
Public OK As Boolean
Private Loaded As Boolean
Private Sub cmdCancel_Click()
    OK = False
    Hide
End Sub

Private Sub cmdOK_Click()
    Dim Temp As New LocalDate
```

```
    Temp.Value = cboStartDate
    If Err.Number = 13 Then
        MsgBox "Please Specify Start Date as DD/MM/YYY"
        Exit Sub
    End If
    Temp.Value = CboEndDate
    If Err.Number = 13 Then
        MsgBox "Please Specify Finish Date as DD/MM/YYY"
        Exit Sub
    End If
    OK = True
    Hide
End Sub

Private Sub Form_Load()
    OK = False
    Dim Temp As New LocalDate
    Temp.Value = Day(Date) & "/" & Month(Date) & "/" & Year(Date)

    If Loaded = False Then
        CboEndDate.Clear
        cboStartDate.Clear
        CboEndDate.Text = Temp.Value
        cboStartDate.Text = Temp.Value
        Temp.Value = Temp.AddDays(-365)
        For I = -365 To 0 Step 1
            CboEndDate.AddItem Temp.Value
            cboStartDate.AddItem Temp.Value
            Temp.Value = Temp.AddDays(1)
        Next
        Loaded = True
    Else
        CboEndDate.Text = Temp.Value
        cboStartDate.Text = Temp.Value
    End If
End Sub
```

B.1.19 FrmShowSimulationRelaibityResult

```
Public Sub ResetControls()

    grdGenerator.Width = 7000
    grdGenerator.Height = 1800
    grdState.Height = 3800
    grdState.Width = 8500
    Dim grdLen As Long, I As Integer

    grdLen = 0
    For I = 0 To grdGenerator.Cols - 1
        grdLen = grdLen + grdGenerator.ColWidth(I)
    Next
    If grdLen + 500 < grdGenerator.Width Then grdGenerator.Width = grdLen + 500

    grdLen = 0
    For I = 0 To grdGenerator.Rows - 1
        grdLen = grdLen + grdGenerator.RowHeight(I)
    Next
    If grdLen + 120 < grdGenerator.Height Then grdGenerator.Height = grdLen + 120
```

```
    If grdGenerator.Height < 1800 Then grdGenerator.Width = grdGenerator.Width - 360

    grdLen = 0
    For I = 0 To grdState.Cols - 1
        grdLen = grdLen + grdState.ColWidth(I)
    Next
    If grdLen + 500 < grdState.Width Then grdState.Width = grdLen + 500

    grdLen = 0
    For I = 0 To grdState.Rows - 1
        grdLen = grdLen + grdState.RowHeight(I)
    Next
    If grdLen + 120 < grdState.Height Then grdState.Height = grdLen + 120
    If grdState.Height < 3800 Then grdState.Width = grdState.Width - 360

    lalProbTable.Top = grdGenerator.Top + grdGenerator.Height + 120
    lalProbTable.Width = grdState.Width
    grdState.Top = lalProbTable.Top + lalProbTable.Height

    lalTotalFlow.Top = grdState.Top + grdState.Height + 120
    lalTotalFlow.Left = grdState.Left
    lalTotalFlow.Width = grdState.Width / 2
    lalExepectedLossE.Left = lalTotalFlow.Left + lalTotalFlow.Width
    lalExepectedLossE.Width = lalTotalFlow.Width
    lalExepectedLossE.Top = lalTotalFlow.Top

    Height = lalExepectedLossE.Top + lalExepectedLossE.Height + 500
    Width = grdState.Width + 335

    cmdOK.Left = grdState.Left + grdState.Width - cmdOK.Width
    cmdPrint.Left = cmdOK.Left

End Sub

Private Sub cmdOK_Click()
    Hide
    Unload Me
End Sub

Private Sub cmdPrint_Click()

    'MS Sans Serif ,Size=8
    Dim W As Long 'Width
    Dim I As Integer, J As Long, Landscape As Boolean
    W = 0
    For I = 0 To grdState.Cols - 1
        W = W + grdState.ColWidth(I)
    Next
    If Printer.Width < W + 1000 Then
        Landscape = True
        Printer.Orientation = vbPRORLandscape
    Else
        Landscape = False
        Printer.Orientation = vbPRORPortrait
        'Printer.Orientation = vbPROR
    End If
    Dim LeftMargin As Integer
    LeftMargin = 800
    Printer.FontName = "MS Sans Serif"
```

```
    Printer.FontBold = False
    Printer.FontItalic = False
    Printer.CurrentY = LeftMargin
    Printer.CurrentX = LeftMargin
    Printer.FontSize = 16
    Printer.Print Caption

    Printer.CurrentX = LeftMargin
    Printer.FontSize = 12
    Printer.Print "Generators Information Table"
    Printer.FontSize = 8

    Printer.FontSize = 8
    Printer.CurrentY = Printer.CurrentY + 60
    PrintGrid grdGenerator, CLng(LeftMargin)

    Printer.FontSize = 12
    Printer.CurrentX = LeftMargin
    Printer.Print lalProbTable

    Printer.FontSize = 8
    Printer.CurrentY = Printer.CurrentY + 60
    PrintGrid grdState, CLng(LeftMargin)

    Printer.FontSize = 12
    Printer.CurrentX = LeftMargin
    Printer.Print lalTotalFlow

    Printer.CurrentX = LeftMargin
    Printer.Print lalExepectedLossE

    Printer.EndDoc

End Sub
```

B.1.20 FrmSimulationLoadFlowResult

```
Private Sub cmdPrint_Click()
    'Printing
    Dim TempY As Long, TempX As Long
    Dim TopM As Long
    Printer.FontSize = 8
    Printer.CurrentX = 500
    Printer.CurrentY = 400
    Printer.Print Caption
    Printer.CurrentX = 500
    Printer.Print Label1.Caption

    TopM = grdA.Top + 500
    TempX = 0
    For I = 0 To grdA.Cols - 1
       TempX = TempX + grdA.ColWidth(I)
    Next
    TempY = 0
    For I = 0 To grdA.Rows - 1
       TempY = TempY + grdA.RowHeight(I) * 0.9
    Next
```

```
    'Printing MatrixA

    Printer.Line (grdA.Left - 30 + 500, TopM - 60)-(grdA.Left + 80 + 500, TopM - 60)
    Printer.Line (grdA.Left - 30 + 500, TopM - 60)-(grdA.Left - 30 + 500, TopM + TempY +
60)
    Printer.Line (grdA.Left - 30 + 500, TopM + TempY + 60)-(grdA.Left + 80 + 500, TopM +
TempY + 60)
    Printer.Line (grdA.Left + TempX + 30 + 500, TopM - 60)-(grdA.Left + TempX - 80 +
500, TopM - 60)
    Printer.Line (grdA.Left + TempX + 30 + 500, TopM - 60)-(grdA.Left + TempX + 30 +
500, TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX + 30 + 500, TopM + TempY + 60)-(grdA.Left + TempX
- 80 + 500, TopM + TempY + 60)
    Printer.CurrentY = TopM
    PrintGrid grdA, grdA.Left + 500, False
    'TempY = Printer.CurrentY

    'Printing MatrixX
    TempX = TempX + 140
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM - 60)-(grdA.Left + TempX + 80 +
500, TopM - 60)
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM - 60)-(grdA.Left + TempX - 30 + 500,
TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM + TempY + 60)-(grdA.Left + TempX
+ 80 + 500, TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM - 60)-(grdA.Left
+ TempX + grdB.ColWidth(0) - 80 + 500, TopM - 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM - 60)-(grdA.Left
+ TempX + grdB.ColWidth(0) + 30 + 500, TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM + TempY + 60)-
(grdA.Left + TempX + grdB.ColWidth(0) - 80 + 500, TopM + TempY + 60)
    Printer.CurrentY = TopM
    PrintGrid grdX, grdA.Left + TempX + 500, False

    Printer.CurrentY = Printer.CurrentY + 500
    PrintGrid MSFlexGrid1, grdA.Left + 500

    TempX = TempX + grdB.ColWidth(0) + 140 + 220
        'Printing MatrixC
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM - 60)-(grdA.Left + TempX + 80 +
500, TopM - 60)
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM - 60)-(grdA.Left + TempX - 30 + 500,
TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX - 30 + 500, TopM + TempY + 60)-(grdA.Left + TempX
+ 80 + 500, TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM - 60)-(grdA.Left
+ TempX + grdB.ColWidth(0) - 80 + 500, TopM - 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM - 60)-(grdA.Left
+ TempX + grdB.ColWidth(0) + 30 + 500, TopM + TempY + 60)
    Printer.Line (grdA.Left + TempX + grdB.ColWidth(0) + 30 + 500, TopM + TempY + 60)-
(grdA.Left + TempX + grdB.ColWidth(0) - 80 + 500, TopM + TempY + 60)
    Printer.CurrentY = TopM + TempY / 2 - 90
    TempX = grdA.Left + TempX + 500 - 240
    Printer.CurrentX = TempX
    Printer.Print "="

    TempX = TempX - grdA.Left - 500 + 240
    Printer.CurrentY = TopM
    PrintGrid grdB, grdA.Left + TempX + 500, False
```

```
    Printer.EndDoc

End Sub

Private Sub Command1_Click()
    Hide
End Sub

Public Function ShowDisplaySimulations(A As Matrix, B As Matrix)
    Const HandW = 6000
    Label1 = "Linear System of " & A.TotalRows & " equation(s) of " & A.TotalRows & "
variables in Matrix Form"
    grdA.Height = HandW
    grdA.Width = HandW
    grdB.Height = grdA.Height
    grdX.Height = grdB.Height
    grdA.Rows = A.TotalRows
    grdA.Cols = A.TotalCols
    grdB.Rows = grdA.Rows
    grdX.Rows = grdA.Rows
    grdB.Cols = 1
    Label2.FontSize = 16
    Dim I As Integer, J As Integer
    For I = 0 To grdA.Cols - 2
        grdA.ColWidth(I) = 300
        grdA.RowHeight(I) = 300
        grdX.RowHeight(I) = 300
        grdB.RowHeight(I) = 300
    Next
        grdA.ColWidth(grdA.Cols - 1) = 20
        grdA.RowHeight(grdA.Cols - 1) = 200
        grdX.RowHeight(grdA.Cols - 1) = 200
        grdB.RowHeight(grdA.Cols - 1) = 200
        grdB.ColWidth(0) = 400
        grdX.ColWidth(0) = 400
    For I = 0 To A.TotalRows - 1
        For J = 0 To A.TotalCols - 1
            Label2 = A.GetValue(I + 1, J + 1)
            If Label2.Width - 20 > grdA.ColWidth(J) Then grdA.ColWidth(J) = Label2.Width +
20
            grdA.Row = I
            grdA.col = J
            grdA.Text = Label2
            grdA.CellAlignment = 3
        Next
        Label2 = B.GetValue(I + 1, 1)
        If Label2.Width - 20 > grdB.ColWidth(0) Then grdB.ColWidth(0) = Label2.Width + 20
        grdB.Row = I
        grdB.col = 0
        grdB.Text = Label2
        grdB.CellAlignment = 3
        Label2 = "Phase" & I
        If Label2.Width - 20 > grdX.ColWidth(0) Then grdX.ColWidth(0) = Label2.Width + 20
        grdX.TextMatrix(I, 0) = "Phase" & I
        grdX.col = 0
        grdX.Row = I
        grdX.CellAlignment = 3
    Next
```

```
    Dim grdLen As Integer
    grdLen = 0
    For I = 0 To grdA.Rows - 1
       grdLen = grdLen + grdA.RowHeight(I)
    Next
    grdA.Height = grdLen + 120
    If grdLen > HandW Then grdA.Height = HandW
    grdLen = 0
    For I = 0 To grdA.Cols - 1
       grdLen = grdLen + grdA.ColWidth(I)
    Next
    If grdLen < HandW + 120 Then
       grdA.Width = grdLen + 120
    Else
       grdA.ColWidth(A.TotalCols - 1) = 400
       grdA.Width = HandW + 120
    End If

    SetLines grdA, Line1, Line2, Line3, Line4, Line5, Line6

    grdX.Height = grdA.Height
    grdX.Width = grdX.ColWidth(0) + 300
    grdX.Left = grdA.Left + grdA.Width + 300
    SetLines grdX, Line7, Line8, Line9, Line10, Line11, Line12

    Label3.Left = grdX.Left + grdX.Width + 220
    Label3.Top = grdX.Top + grdX.Height / 2 - Label3.Height / 2

    grdB.Height = grdA.Height
    grdB.Width = grdB.ColWidth(0) + 300
    grdB.Left = Label3.Left + Label3.Width + 320
    SetLines grdB, Line13, Line14, Line15, Line16, Line17, Line18

    If A.TotalCols <= 2 Then
       Label1.Top = 20
       If A.TotalCols = 1 Then Command1.Width = 1215 * 0.75
    Else
       Label1.Top = 120
       Command1.Width = 1215
    End If
    Label1.Width = Line16.X1
    Width = Label1.Width + Label1.Left * 2
    MSFlexGrid1.Top = grdA.Top + grdA.Height + 200
    If grdA.Width > 2200 Then MSFlexGrid1.Width = grdA.Width
    Height = MSFlexGrid1.Top + MSFlexGrid1.Height + 420
    Command1.Left = ScaleWidth - Command1.Width - 120
    Command1.Top = ScaleHeight - Command1.Height - 120
    cmdPrint.Left = Command1.Left
    cmdPrint.Width = Command1.Width
    cmdPrint.Top = Command1.Top - cmdPrint.Height - 120

End Function

Private Sub SetLines(Gdr As MSFlexGrid, LineLeft As Line, LineLeftTop As Line,
LineLeftDown As Line, LineRight As Line, LineRightTop As Line, LineRightDown As Line)

    LineLeft.X1 = Gdr.Left - 80
    LineLeft.X2 = LineLeft.X1
```

```
    LineLeft.Y1 = Gdr.Top - 40
    LineLeft.Y2 = Gdr.Top + Gdr.Height + 20

    LineLeftTop.X1 = LineLeft.X1
    LineLeftTop.Y1 = LineLeft.Y1
    LineLeftTop.Y2 = LineLeft.Y1
    LineLeftTop.X2 = LineLeft.X1 + 70

    LineLeftDown.X1 = LineLeft.X2
    LineLeftDown.Y1 = LineLeft.Y2
    LineLeftDown.Y2 = LineLeft.Y2
    LineLeftDown.X2 = LineLeft.X2 + 70

    LineRight.X1 = Gdr.Left + Gdr.Width + 40
    LineRight.X2 = LineRight.X1
    LineRight.Y1 = Gdr.Top - 40
    LineRight.Y2 = Gdr.Top + Gdr.Height + 20

    LineRightTop.X1 = LineRight.X1
    LineRightTop.Y1 = LineRight.Y1
    LineRightTop.Y2 = LineRight.Y1
    LineRightTop.X2 = LineRight.X1 - 70

    LineRightDown.X1 = LineRight.X2
    LineRightDown.Y1 = LineRight.Y2
    LineRightDown.Y2 = LineRight.Y2
    LineRightDown.X2 = LineRight.X2 - 70

End Sub
```

B.1.21 FrmTransformer

```
'=========================================================
====================================
'=========================================================
====================================
'   THE GOALS OF THIS FORM IS
'   1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF TRANSFORMER'S BASIC
PARAMETERS
'=========================================================
====================================
'=========================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitBreakers"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitBreakers"
Option Explicit
Private MyMaitinadate As New LocalDate, ChangeMDate As Boolean

Private Sub Form_Activate()

    If txtDateInstalled = "__/__/____" Or txtDateInstalled = "" Then
        txtDateInstalled = Format(Day(Date), "00") & "/" & Format(Month(Date), "00") & "/"
& Year(Date)
        MyMaitinadate.Value = txtDateInstalled
    Else
        MyMaitinadate.Value = txtNextMaintenanceDate
    End If
    ChangeMDate = True
```

```
End Sub

Private Sub Form_Unload(Cancel As Integer)
   ChangeMDate = False
End Sub

Private Sub txtDateInstalled_Change()

   'The following change will take place when
   'InstalledDate is not empty AND current form is displayed on screen
   If txtDateInstalled <> "__/__/____" And ChangeMDate Then
      MyMaitinadate.Value = txtDateInstalled
      Call txtMaintenanceInterval_Change
   End If

End Sub

Private Sub txtMaintenanceInterval_Change()

   'On Error Resume Next
   If ChangeMDate Then txtNextMaintenanceDate =
MyMaitinadate.AddDays(CLng(Val(txtMaintenanceInterval)))

End Sub

Private Sub cmdCancel_Click()

   Hide
   ChangeMDate = False

End Sub

Private Sub cmdSAVE_Click()

   SaveFormData Me, UpdateRecord
   Hide
   ChangeMDate = False
End Sub

Private Sub CmdShowDate_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtDateInstalled = Temp 'GETING DATE BY DISPLAYING A
CALENDER

End Sub

Private Sub CmdShowDate2_Click()
   Dim Temp As String
   Temp = GetDate(Me)
   If Temp <> "" Then txtNextMaintenanceDate = Temp 'GETING DATE BY
DISPLAYING A CALENDER

End Sub
```

B.1.22 FrmTransformerFailure

```
'=========================================================
====================================
'=========================================================
====================================
'   THE GOALS OF THIS FORM ARE
'    1.  PROVIDE THE USER INTERFACE TO GET INFORMATION OF TRANSFORMERS'
FAILURE
'   2.  SHOW SPECIFIC RECORDS RELATED TO A TRANSFORMER OR ALL
'=========================================================
====================================
'=========================================================
====================================
'THIS FORM IS SIMILAR TO "FrmCircuitbreakerFailure"
'PLEASE LOOK AT COMMENTS IN FORM "FrmCircuitbreakerFailure"
Private Sub CmdClose_Click()

    Hide

End Sub

Private Sub CmdMoveFirst_Click()

    If optAll Then
        rsForAll.MoveFirst
    Else
        rsForAll.FindFirst "Transformersid='" & txtTransformersID & "'"
    End If
    LoadFormData Me

End Sub

Private Sub CmdMoveLast_Click()
    If optAll Then
        rsForAll.MoveLast
    Else
        rsForAll.FindLast "Transformersid='" & txtTransformersID & "'"
    End If
    LoadFormData Me
End Sub

Private Sub CmdMoveNext_Click()

    If optAll Then
        rsForAll.MoveNext
        If rsForAll.EOF Then
            MsgBox "This is Last record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindNext "Transformersid='" & txtTransformersID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is Last record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub
```

```
Private Sub CmdMovePrevious_Click()

    If optAll Then
        rsForAll.MovePrevious
        If rsForAll.BOF Then
            MsgBox "This is first record", vbInformation
            rsForAll.MoveLast
        End If
    Else
        rsForAll.FindPrevious "Transformersid='" & txtTransformersID & "'"
        If rsForAll.NoMatch = True Then
            MsgBox "This is first record", vbInformation
        End If
    End If
    LoadFormData Me

End Sub

Private Sub CmdUpdate_Click()

    SaveFormData Me, UpdateRecord
    MsgBox "Record is updated", vbOKOnly + vbInformation, "Record Updation"

End Sub

Private Sub CmdShowDate_Click()

    txtFailureDate = GetDate(Me)

End Sub
```

B.1.23 FrmWait

```
Private TotalSteps As Integer

Private Sub lalCounter_Change()
    ProgressBar1.Value = lalCounter
    'If ProgressBar1.Value = ProgressBar1.Max Then Hide
    Hide
End Sub

Private Sub lalTotal_Change()
    TotalSteps = lalTotal
    ProgressBar1.Max = TotalSteps
    ProgressBar1.Min = 0
End Sub
```

B.1.24 Help

```
Private Sub CmdClose_Click()
    Unload Me
    Call Load(Frmlogin)
    Call Frmlogin.Show
End Sub
```

B.1.25 MainWindow

```
'=========================================================
====================================
'=========================================================
====================================
'  THE GOALS OF THIS FORM ARE
'  1.  PROVIDE THE USER STANDARD APPLICATION VIEW
'   2.  TOOLBAR FOR CREATING FOLLOWING CONTROLS ON DEGNING SPACE TO MAKE
CUIRCUITS
'      A.  TRANFORMER
'      B.  GENERATOR
'      C.  BUSBAR LINE
'      D.  TRANSMISSION LINE
'      E.  FEEDER
'  3.  SENDING INFORMATION TO fFrmMain OF THE CONTROL TO BE CREATED
'  4.  PROVIDEING fFrmMain THE POPUP MENU WITH
'         A.  AddNew Failure Parameters
'         B.  Edit Basic Parameters
'         C.  Edit Failure Parameters
'  5.  SAVING CUIRCUIT MODEL FILE WITH ALL OF THE CUIRCUT INFORMATION AS .MDB
'
'  -->>  THIS FORM COMMUNICATE WITH THE MAINWINDOW FORM BY MEAN OF
'         1.  MODULE "modPublicProcedures"
'         2.  AND SOME PUBLIC VARIABLES DEFINED IN MODULE "modCommonVariables"
'=========================================================
====================================
'=========================================================
====================================

Option Explicit
Private Terminate As Boolean

Private Sub MDIForm_Load()
   CloseApp = 0
   Frmlogin.Show
   Frmlogin.Refresh
   Dim Temp
   Temp = Second(Time) + 2
   If Temp >= 60 Then Temp = 1
   Dim I As Integer
   Do While Temp > Second(Time)
      DoEvents
   Loop
   Frmlogin.Hide
   Unload Frmlogin
   frmSimulationLoadFlowResult.Hide
   TimerStart.Enabled = True
End Sub

Private Sub MDIForm_Unload(Cancel As Integer)
   If CloseApp = -1 Then
      Cancel = 1
   ElseIf CloseApp = 1 Then
      'do nothing just exiting
      'End
   Else
      Cancel = 1
      TimerGotoExit.Enabled = True
```

```
    End If
End Sub

Private Sub mnu_ApplyFailureCondition_Click()
    fMainForm(ActiveFormID).ApplyFailureCondition
End Sub

Private Sub mnu_RemoveFailureCondition_Click()
    fMainForm(ActiveFormID).RemoveFailureCondition
End Sub

Private Sub MnuApplication_Click()

    MnuApplication.Checked = Not MnuApplication.Checked
    Toolbar3.Visible = MnuApplication.Checked

End Sub

Private Sub Mnucopy_Click()
    fMainForm(ActiveFormID).Copy
End Sub

Private Sub MnuCopyControl_Click()
    Call Mnucopy_Click
End Sub

Private Sub Mnucut_Click()
    fMainForm(ActiveFormID).Cut
End Sub

Private Sub mnuCutControl_Click()
    Call Mnucut_Click
End Sub

Private Sub Mnudelete_Click()
    fMainForm(ActiveFormID).DeleteItem
End Sub

Private Sub mnuFileSaveAs_Click()

    fMainForm(ActiveFormID).MyFileName = ""
    fMainForm(ActiveFormID).SaveFile

End Sub

Private Sub MnuMaintenance_Click()
    Dim DayWeek As Integer, PDate As Date
    DayWeek = Weekday(Date)
    PDate = Date - DayWeek + 1

    mnuThisYear.Caption = "&Year   " & Format(Date, " YYYY")
    mnuThisMonth.Caption = "Month  " & "&" & Format(Date, "Mmmm YYYY")
    mnuthisweek.Caption = "Week   " & Format(PDate, "D MMM, YYYY") & " &To " &
Format(PDate + 6, "D MMM, YYYY")
    mnuToday.Caption = "Today   " & "&" & Format(Date, "D MMMM, YYYY")

    mnuNext6days.Caption = "6 Days        " & Format(Date, "D MMM, YYYY") & " &To " &
Format(Date + 6, "D MMM, YYYY")
```

```
    mnuNext30days.Caption = "30 Days       " & Format(Date, "D MMM, YYYY") & " &To " &
Format(Date + 30, "D MMM, YYYY")
    mnuNext365days.Caption = "365 Days      " & Format(Date, "D MMM, YYYY") & " &To "
& Format(Date + 365, "D MMM, YYYY")

End Sub

Private Sub mnuNext30days_Click()
    ShowMaintenanceShudule 4
End Sub

Private Sub mnuNext365days_Click()
    ShowMaintenanceShudule 5
End Sub

Private Sub mnuNext6days_Click()
    ShowMaintenanceShudule 3
End Sub

Private Sub Mnupaste_Click()
    fMainForm(ActiveFormID).Paste
End Sub

Private Sub Mnustandard_Click()
    Mnuselectall.Checked = Not Mnuselectall.Checked
    Toolbar1.Visible = Mnuselectall.Checked
End Sub

Private Sub MnuStandardToolBar_Click()
    MnuStandardToolBar.Checked = Not MnuStandardToolBar.Checked
    Toolbar1.Visible = MnuStandardToolBar.Checked
End Sub

Private Sub mnuThisMonth_Click()
    ShowMaintenanceShudule 2
End Sub

Private Sub mnuthisweek_Click()
    ShowMaintenanceShudule 1
End Sub

Private Sub mnuThisYear_Click()
    ShowMaintenanceShudule 6
End Sub

Private Sub mnuToday_Click()
    ShowMaintenanceShudule 0
End Sub

Private Sub MnuToolBox_Click()

    MnuToolBox.Checked = Not MnuToolBox.Checked
    Toolbar2.Visible = MnuToolBox.Checked
    Toolbar4.Visible = MnuToolBox.Checked

End Sub

Private Sub TimerGotoExit_Timer()
    TimerGotoExit.Enabled = False
```

```
    Call Mnuexit_Click
End Sub

Private Sub TimerStart_Timer()
    Main    'Calling main for all initail steps
    TimerStart.Enabled = False
End Sub

Private Sub Toolbar2_ButtonClick(ByVal button As ComctlLib.button)
    'The folowing code providet he facility that
    '   1.  If user click on Generator control --> It is seemd to be pressed
    '   2.  If user click on Generator control agian --> It is seemd now to be unpressed
    If button.MixedState = True Then    'Coontrol is Pressed
        ButtonKey = ""  'To make it unpressed
        DisableAllButton
    Else
        DisableAllButton
        button.MixedState = True
        ButtonKey = button.Key
    End If

End Sub

'THE FOLLOWING(#-#-#-)MENUS ARE USED AS A POPUP MENU
'PLEASE LOOK THE LAST PARAMETER OF THE PROCEDURE "OpenMyInfoForm"
'#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-
#-#-#-#-#-#-#-#-
Private Sub mnuAddNewFailureParameters_Click()
    OpenMyInfoForm                                   fMainForm(ActiveFormID).CurrentControlID,
fMainForm(ActiveFormID).DB, ShowFailuresAddNew
End Sub
Private Sub mnuBasicParameters_Click()
    OpenMyInfoForm                                   fMainForm(ActiveFormID).CurrentControlID,
fMainForm(ActiveFormID).DB, ShowCharacteristics
End Sub
Private Sub mnuEditFailureParameters_Click()
    'THIS PROCEDURE AUTOMATICALLY OPEN THE FEEDER FAILURE FORM FOR EDIT
    OpenMyInfoForm                                   fMainForm(ActiveFormID).CurrentControlID,
fMainForm(ActiveFormID).DB, ShowFailuresEdit
End Sub
'#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-
#-#-#-#-#-#-#-#-


Private Sub Mnuexit_Click()
    On Error Resume Next    'TO AVOID ERROR WHILE TERMINATING
    CloseApp = 1
    Unload Me
    If CloseApp = 1 Then End
End Sub

Private Sub mnuFileNew_Click()

    LoadNewDoc

End Sub

Private Sub mnuFileOpen_Click()
```

```
    MainWindow.Commondialog.Filter = "Drawing Database .mdb|*.mdb"
    Commondialog.FileName = ""
    Commondialog.ShowOpen
    If Trim(Commondialog.FileName) <> "" Then LoadNewDoc Commondialog.FileName

End Sub

Private Sub mnuFileSave_Click()

    fMainForm(ActiveFormID).SaveFile

End Sub

Private Sub Mnuprint_Click()

    'fMainForm(ActiveFormID).HScroll.Value = 0
    'fMainForm(ActiveFormID).VScroll.Value = 0
    Dim ctrl As Control, I As Long
        frmPrintPreview.PicPreview.BackColor = RGB(255, 255, 255)
        '.AutoRedraw = True
    With fMainForm(ActiveFormID)
            frmPrintPreview.PicPreview.Width = .PicDrawing.Width + 120
            frmPrintPreview.PicPreview.Height = .PicDrawing.Height + 120
            frmPrintPreview.PicPreview.DrawWidth = 1
            'For I = .PicDrawing.Top To .PicDrawing.Width Step 144
            '    If I Mod 5 = 0 Then
            '        frmPrintPreview.PicPreview.Line (0, I)-(.PicScaleVertical.Width / 2, I)
            '    Else
            '        frmPrintPreview.PicPreview.Line (0, I)-(.PicScaleVertical.Width / 4, I)
            '    End If
            'Next
            'For I = 0 To .PicDrawing.Width Step 1440
            '    frmPrintPreview.PicPreview.CurrentX = 0
            '    frmPrintPreview.PicPreview.CurrentY = I  'frmPrintPreview.PicPreview.Height / 3
            '    frmPrintPreview.PicPreview.Print Int((I - .PicDrawing.Top) / 1440)
            '    frmPrintPreview.PicPreview.Line (0, I)-(.PicScaleVertical.Width, I), RGB(200, 50,
50)
            'Next
            'For I = .PicDrawing.Left To .PicDrawing.Width Step 144
            '    If (I - .PicDrawing.Left) Mod 5 = 0 Then
            '        frmPrintPreview.PicPreview.Line (I , 0)-(I , .PicScaleHorizental.Height / 2)
            '    Else
            '        frmPrintPreview.PicPreview.Line (I , 0)-(I , .PicScaleHorizental.Height / 4)
            '    End If
            'Next
            'For I = .PicDrawing.Left To .PicDrawing.Width Step 1440
            '    frmPrintPreview.PicPreview.CurrentX = I
            '    frmPrintPreview.PicPreview.CurrentY = 0 'frmPrintPreview.PicPreview.Height / 3
            '    frmPrintPreview.PicPreview.Print Int((I - .PicDrawing.Left) / 1440)
            '    frmPrintPreview.PicPreview.Line (I, 0)-(I , .PicScaleHorizental.Height), RGB(200,
50, 50)
            'Next
        For Each ctrl In fMainForm(ActiveFormID)
            frmPrintPreview.PicPreview.ForeColor = 0
            If InStr(ctrl.Name, "img") > 0 Then
                frmPrintPreview.PicPreview.PaintPicture ctrl.Picture, ctrl.Left, ctrl.Top
            ElseIf InStr(LCase(ctrl.Name), "lal") > 0 Then
                frmPrintPreview.PicPreview.CurrentX = ctrl.Left
                frmPrintPreview.PicPreview.CurrentY = ctrl.Top
```

```
            frmPrintPreview.PicPreview.Print ctrl.Caption
        ElseIf InStr(LCase(ctrl.Name), "line") = 1 Then
            frmPrintPreview.PicPreview.DrawWidth = 2
            frmPrintPreview.PicPreview.ForeColor = ctrl.BorderColor
            frmPrintPreview.PicPreview.Line (ctrl.X1, ctrl.Y1)-(ctrl.X2, ctrl.Y2)
        ElseIf InStr(LCase(ctrl.Name), "shape") > 0 Then
            frmPrintPreview.PicPreview.DrawWidth = 1
            frmPrintPreview.PicPreview.ForeColor = ctrl.BorderColor
            frmPrintPreview.PicPreview.PSet (ctrl.Left, ctrl.Top)
        End If
    Next
End With
'Form1.Picture = fMainForm(ActiveFormID).PicDrawing.Picture
'.AutoRedraw = 0


Printer.Orientation = vbPRORLandscape
'Now printing on paper(s)
With frmPrintPreview.PicPreview
    Dim J As Long, H As Long, W As Long
    Dim Px As Long, Py As Long      'To hold Previous point
    H = Printer.Height - 1440   'geting Page Height
    W = Printer.Width - 1440    'geting Page Width
    If .Height - 120 > H Or .Width - 120 > W Then
        For I = 0 To .Width Step W
            For J = 0 To .Height Step H
                If J + H < .Height Then
                    If I + W < .Width Then
                        Printer.PaintPicture .Image, 720, 720, , , I, J, W + 60, H + 160
                        PrintScale I, W + 160, J, H + 160
                    Else
                        Printer.PaintPicture .Image, 720, 720, , , I, J, .Width - I, H + 160
                        PrintScale I, .Width - I, J, H + 160
                    End If
                Else
                    If I + W < .Width Then
                        Printer.PaintPicture .Image, 720, 720, , , I, J, W + 160, .Height - J
                        PrintScale I, W + 160, J, .Height - J
                    Else
                        Printer.PaintPicture .Image, 720, 720, , , I, J, .Width - I, .Height - J
                        PrintScale I, .Width - I, J, .Height - J
                    End If
                End If
                Printer.NewPage
                Printer.Orientation = vbPRORLandscape
            Next
        Next
    Else
        Printer.PaintPicture .Image, 720, 720
        PrintScale 0, .Width, 0, .Height
    End If
End With
Printer.EndDoc
'frmPrintPreview.Show

'Printer.PaintPicture frmPrintPreview.PicPreview.Image, 0, 0
'fMainForm(ActiveFormID).PrintForm
'Printer.Orientation = vbPRORPortrait
```

```
End Sub

Private Sub PrintScale(StartingX As Long, LenghtX As Long, StartingY As Long, LenghtY As
Long)

    Dim I As Long, J As Long
    For I = 720 To LenghtX + 720 Step 144
        If (I + StartingX - 720) Mod 5 = 0 Then
            Printer.Line (I, 360)-(I, 540)
        Else
            Printer.Line (I, 360)-(I, 360 + 360 / 4)
        End If
    Next
    For I = 720 To LenghtX + 720
        If (I - 720 + StartingX) Mod 1440 = 0 Then
            Printer.CurrentX = I
            Printer.CurrentY = 360 'printer.Height / 3
            Printer.Print Int((I - 720 + StartingX) / 1440)
            Printer.Line (I, 360)-(I, 720), RGB(200, 50, 50)
        End If
    Next

    'Vertical Scale
    For I = 720 To LenghtY + 720 Step 144
        If (I + StartingY - 720) Mod 5 = 0 Then
            Printer.Line (360, I)-(540, I)
        Else
            Printer.Line (360, I)-(360 + 360 / 4, I)
        End If
    Next
    For I = 720 To LenghtY + 720
        If (I - 720 + StartingY) Mod 1440 = 0 Then
            Printer.CurrentX = 360
            Printer.CurrentY = I 'printer.Height / 3
            Printer.Print Int((I - 720 + StartingY) / 1440)
            Printer.Line (360, I)-(720, I), RGB(200, 50, 50)
        End If
    Next


End Sub

'REST OF THE CODE FOR THIS FORM IS CRARETED LATER (BY YOU)
Private Sub Toolbar1_ButtonClick(ByVal button As ComctlLib.button)
    'On Error Resume Next
    Select Case button.Key
        Case "New"
            Call mnuFileNew_Click
        Case "Open"
            Call mnuFileOpen_Click
        Case "Save"
            Call mnuFileSave_Click
        Case "Print"
            Call Mnuprint_Click
        Case "Cut"
            fMainForm(ActiveFormID).Cut
        Case "Copy"
            fMainForm(ActiveFormID).Copy
        Case "Paste"
```

```
            fMainForm(ActiveFormID).Paste
        Case "Delete"
            fMainForm(ActiveFormID).DeleteItem
    End Select
End Sub


Private Sub Toolbar3_ButtonClick(ByVal button As ComctlLib.button)

Dim I As Integer

    If button.Key = "Play" Then
        If button.MixedState = True Then    'Control is Pressed
            ButtonKey = ""  'To make it unpressed
            For I = 1 To 8
                MainWindow.Toolbar3.Buttons(I).MixedState = False
            Next
        Else
            For I = 1 To 8
                MainWindow.Toolbar3.Buttons(I).MixedState = False
            Next
            button.MixedState = True
            ButtonKey = button.Key
        End If

        fMainForm(ActiveFormID).DisplayFlow
    ElseIf button.Key = "Stop" Then
            For I = 1 To 8
                MainWindow.Toolbar3.Buttons(I).MixedState = False
            Next
        fMainForm(ActiveFormID).TimerSimulationFlow.Enabled = False
        fMainForm(ActiveFormID).PicDrawing.Cls
        fMainForm(ActiveFormID).clearlal
    ElseIf button.Key = "Pause" Then
        fMainForm(ActiveFormID).TimerSimulationFlow.Enabled = False
    ElseIf button.Key = "NodeToNode" Then
        fMainForm(ActiveFormID).N_to_N_Detection
    ElseIf button.Key = "LoadFlowCalculations" Then
        fMainForm(ActiveFormID).Simulate_LoadFlow
    ElseIf button.Key = "RelaibilityCaculations" Then
        fMainForm(ActiveFormID).Simulate_Reliability
    ElseIf button.Key = "performance" Then
        fMainForm(ActiveFormID).Simulate_PerformanceIndicesCal
    ElseIf button.Key = "AC" Then
            If button.MixedState = True Then    'Control is Pressed
            ButtonKey = ""  'To make it unpressed
            For I = 1 To 8
                MainWindow.Toolbar3.Buttons(I).MixedState = False
            Next
        Else
            For I = 1 To 8
                MainWindow.Toolbar3.Buttons(I).MixedState = False
            Next
            button.MixedState = True
            ButtonKey = button.Key
        End If
        fMainForm(ActiveFormID).Combi
    End If
```

```
End Sub

Private Sub Toolbar4_ButtonClick(ByVal button As ComctlLib.button)

   If button.Key = "Rotate" Then
      fMainForm(ActiveFormID).Rotate
   End If

End Sub

'"New
Private Sub mnuAddDelete_Click()

Dim User As String
Dim pw As String
Dim db1 As Database
Dim record1 As Recordset
Dim record2 As Recordset
Dim record3 As Recordset

Dim dbpath As String
Dim sPassword As String
Dim SQL1 As String
Dim SQL2 As String
Dim SQL3 As String
Dim wrong As Integer

wrong = 0
User = LogIn()
pw = LogIn1()
'MsgBox "username =" & user
'MsgBox "password =" & pw

dbpath = "\db1.mdb"
Set db1 = OpenDatabase(App.Path & dbpath, dbDriverComplete, False,
";DATABASE=;VID=;PWD=" & sPassword & ";DSN=")
SQL1 = "SELECT * FROM userdata"
SQL2 = "SELECT Privilege FROM userdata where UserID = '" + User + "' and Password = '"
+ pw + "'"
SQL3 = "SELECT count(*) FROM userdata"

Set record1 = db1.OpenRecordset(SQL1, dbOpenDynaset)
Set record2 = db1.OpenRecordset(SQL2, dbOpenDynaset)
Set record3 = db1.OpenRecordset(SQL3, dbOpenDynaset)

With record1
Do While Not record1.EOF
If !UserID = User Or UCase(!UserID) = User Then
   If !Password = pw Then
      With record2
      If !Privilege = 1 Or UCase(!Privilege) = 1 Then
         Call Load(FrmAdd)
         Call FrmAdd.Show
      Else
         wrong = wrong + 1
      End If

      End With
      record2.Close
```

```
    Else
        wrong = wrong + 1
    End If
Else
    wrong = wrong + 1
End If
.MoveNext
Loop
End With
'MsgBox "wr" & wrong
'MsgBox "re" & record1.RecordCount
If wrong = record1.RecordCount Then
MsgBox "You don't have the Privilege to enter this section.", vbCritical
End If
record1.Close
'    Set db10 = OpenDatabase(dbpath)
'    Set Res = db10.OpenRecordset("userdata")

'For I = 1 To Res.RecordCount
'    userid = Res.Fields("UserID")
'    MsgBox "userid" & I & "=" & userid
 '   Password = Res.Fields("Password")
 '   MsgBox "password" & I & "=" & Password
 '   Level = Res.Fields("Level")
 '   MsgBox "level" & I & "=" & Level
'    If user = userid And pw = Password Then
'        If Level = 0 Then
'            Unload Me
 '       ElseIf Level = 1 Then
'            Call Load(FrmAdd)
'            Call FrmAdd.Show
 '       End If
 '   ElseIf user <> userid Then
 '       wrong = wrong + 1
       'MsgBox "wrong" & wrong
 '   End If
'Res.MoveNext
'Next

'If wrong = Res.RecordCount Then
'MsgBox "You do not have the Privilege to enter this section.", vbCritical
'End If

End Sub
'''additional
Private Sub Form_Load()
Dim dbpath As String
    dbpath = "\db1.mdb"  'stores the database information, such as directory
End Sub
'''New
Public Function LogIn()
   ' Ask the user for a password.
   LogIn = InputBox("UserID")
End Function
'''New
Public Function LogIn1()
   ' Ask the user for a password.
   LogIn1 = InputBox("Password")
End Function
```

```
'''New
Private Sub mnuChange_Click()
    Call Load(FrmChangePW)
    Call FrmChangePW.Show
End Sub
```

B.2 Modules

B.2.1 BusbarAndGenerator

```
Type Point
   X As Double
   Y As Double
End Type
Public Function IsIntersection(M1 As Double, C1 As Double, M2 As Double, C2 As Double)
As Point



End Function

Public Function GeneratorConnectedToBusbar(Gen As Image, Bus As Line) As Point

   ' 15.5 , 13.5
   Dim Bm As Double 'Slope of Busbar
   Dim Bc As Double 'Constant of Busbar Line
   Dim Gc As Double 'Constant of Generator Line Perpendicular to Busbar Line
   Dim Gm As Double

   If Bus.X1 = Bus.X2 Then
      Bm = 9999999999#
      Bc = 0
      Gc = (Gen.Top + 13.5 * 15)
      Gm = 0
   Else
      Bm = (Bus.Y1 - Bus.Y2) / CDbl(Bus.X1 - Bus.X2)
      Bc = Bus.Y2 - Bm * Bus.X2
      If Bm > 0.0000000000001 Then
         Gc = (Gen.Top + 13.5 * 15) - (-1 / Bm) * (Gen.Left + 15.5 * 15)
         Gm = -1 / Bm
      Else
         Gc = 9999999999#
         Gm = 9999999999#
      End If
   End If

   If Bm = Gm Then
      GeneratorConnectedToBusbar.X = -1   'Lines are not intersecting
      GeneratorConnectedToBusbar.Y = -1
      Exit Function
   End If

   If Gm <> 9999999999# And Bm <> 9999999999# Then
      GeneratorConnectedToBusbar.X = -(Gc - Bc) / (Gm - Bm)
      GeneratorConnectedToBusbar.Y = Gm * GeneratorConnectedToBusbar.X + Gc
   ElseIf Gm = 9999999999# Then
      GeneratorConnectedToBusbar.X = Gen.Left + 15.5 * 15
      GeneratorConnectedToBusbar.Y = Bus.Y2 '- 15 '* 2
   ElseIf Bm = 9999999999# Then
      GeneratorConnectedToBusbar.X = Bus.X2
      GeneratorConnectedToBusbar.Y = Gen.Top + 13.5 * 15
   End If
```

```
   If Sqr(((Gen.Top + 13.5 * 15) - GeneratorConnectedToBusbar.Y) ^ 2 + ((Gen.Left +
15.5 * 15) - GeneratorConnectedToBusbar.X) ^ 2) > 19 * 15 Then
      GeneratorConnectedToBusbar.X = -1
      GeneratorConnectedToBusbar.Y = -1
   End If

End Function
```

B.2.2 MatrixOperation

```
Public Function MatMultiply(ByRef A As Matrix, ByRef B As Matrix) As Matrix

   Set MatMultiply = New Matrix
   Dim I As Integer, J As Integer, K As Integer
   If A.TotalCols = B.TotalRows Then
      MatMultiply.Order A.TotalRows, B.TotalCols
      For I = 1 To A.TotalRows
         For J = 1 To B.TotalCols
            For K = 1 To A.TotalCols
            '"MsgBox "inv A (" & I & ", " & K & ") = " & A.GetValue(I, K)
               MatMultiply.ChangeValue(I, J) = MatMultiply.GetValue(I, J) + A.GetValue(I,
K) * B.GetValue(K, J)
            Next
         Next
      Next
   End If

End Function
```

B.2.3 ModCommonVariables

```
Public ActiveFormID As Integer
   'SINCE WE HAVE A FORM ARRAY "fMainForm()" IN MODULE "modPublicProcedures"
   'THIS VARIABLE KEEP THE INFORMATION OF THE INDEX OF fMainForm() OF THE FORM
   'WHICH IS ACTIVE. E.G. IF WE WANT TO ACCESS THE ACTIVE FORM WE SHALL
   'USE fMainForm(ActiveFormID). e.g. TO GET CAPTION fMainForm(ActiveFormID).Caption

Public NewFileName As String
   'THIS IS THE NAME OF FILE TO STORE/OPEN
   'WHEN FORM LOADS, IN LOAD EVENT, IT OPEN THE DATABASE SPECIFIED AS
"NewFileName"
   'THIS IS THE WAY A FORM KNOWS, WHICH FILE IS TO OPEN

Public ButtonKey As String
   'THIS VARIABLE KEEPS THE INFORMATION OF THE TOOLBAR BUTTON'S KEY
   'BY USING THIS VARIABLE AN ACTIVE FORM KNOWS WHICH TOOL TO CREATE
   'THIS VARIABLE IS USED IN FORMS MOUSE DOWN EVENT

Public fMainForm() As FrmMain

Public Type Generators
   ID As String
   RepairTime As Double
End Type
```

```
Public AppPath As String

Public rsForAll As Recordset

Public CutCopyPaste As New MyClipboard

Public Enum ScaleType
   Inchs = 1
   CentiMeter = 2
End Enum

Public CloseApp As Integer
```

B.2.4 ModPublicProcedures

```
'=============================================================
====================================
'=============================================================
====================================
'   THE GOALS OF THIS MODULE ARE
'
'    PROVIDING MainWindow FORM AND ALL NEW CREATED FORMS A MEDIUM TO
COMUNICATE
'
'   1.  CREATING AND STORING REFRENCES FOR NEW FROM MODELS
'   2.  STORING INFORMATION OF ALL FORMS IN THEIR DATABASES
'=============================================================
====================================
'=============================================================
====================================

Private ldMainForm As New LoadUnload    'TO KEEP TRACK THE FORMS REFRENCES
Private lDocumentCount As Long  'USED AS INDEX
Private ModelCounter As Integer 'IT IS INFACT A COUNTER TO ADD THE MODELS TO BE
OPENED IN ONE EXECUTION
'-----------------------------------------------------
Public Enum FormShow    'DEFINED FOR "OpenMyInfoForm"
   ShowCharacteristics = 1     'MEANS TO OPEN BASIC CHRACTERISTICS
   ShowFailuresAddNew = 2      'ADDING NEW VALUE TO FAILURE
   ShowFailuresEdit = 3        'EDITING VALUES OF FAILURE
   ShowNone = 4       'JUST SAVING THE CHANGE IN INFROMATION OF DISPLAY OF A
CONTROL
End Enum
'-----------------------------------------------------
Public Enum RsUpdate    ' DEFINED FOR "SaveFormData"
   UpdateRecord = 1    'TO UPDATE RECORD
   AddNewRecord = 2    'TO ADDNEW RECORD
End Enum
Public Opendimension As Boolean
'-----------------------------------------------------


'MAIN BODY
Sub Main()

   'SendKeys vbKeyControl + vbKeyEscape
   Dim MainWin As New MainWindow
```

```
    Set MainWin = MainWindow
    MainWin.Show
    AppPath = App.Path & "\BackUp\" 'STROING THE PATH OF THE BACKUP FOLDER
    ReDim fMainForm(0)              'INITIALIZING FORM ARRAY
    Set fMainForm(0) = New FrmMain  'CREATING FIRST FORM
    EnableAll False
    'LoadNewDoc  'LOADING NEW DOCUMENT(FORM or MODEL)
End Sub

Public Sub CloseMe(I As Integer)
    'NOT USED BUT MAY BE USEFULL IN NEXT VERSIONS
    'Unload fMainForm(i)

End Sub

Public Sub LoadNewDoc(Optional FileName As String = "")
    'THIS SUB PERFORMS THE FOLLOWING
    'If FileName is given => it opens existing Drawing model
    'If Filename is not given => Insentiate fFrmMain for new Drawing model

    'THIS SUB ALSO COPY THE DDRCMP.sys (INFACT IT IS THE COMPLETE DATABASE FILE
WITH NO ENTERY)
    'IN THIS WAY WE NOT NEED TO CREATE NEW DATABASE USING DAO36.DLL

    lDocumentCount = ldMainForm.NewItemNumber    'GETTING A NUMBER WHERE THE
FORM REFRENCE
            'MUST BE PLACED IN THE ARRAY fFrmMain(). READ CLASS LOADUNLOAD FOR
DETAILS
    If Trim(FileName) <> "" Then
        Dim FSys As New FileSystemObject 'File System Object Used for copying/deleting file
        Opendimension = False
        NewFileName = Left(FileName, Len(FileName) - 4) & ".bk"
        'FOLLOWING LOOP GETING JUST NAME OF THE FILE AND REMOVEING IT'S PATH
FORM THE NAME
        Do While InStr(NewFileName, "\")
            NewFileName = Right(NewFileName, Len(NewFileName) - InStr(NewFileName,
"\"))
        Loop
        'CHECKING THE FILE IS ALRADY OPEN BY THE PROGRAM OR NOT
        If FSys.FileExists(AppPath & NewFileName & ".ldb") = True Then
            MsgBox "File already open", vbOKOnly + vbCritical, "FILE ALREADY OPEN"
            ldMainForm.DeleteAtIndex CInt(lDocumentCount) + 1    'REMOVING UNOCCUPIED
SPACE INDEX
            Exit Sub
        Else
            'COPYING FILE IN THE BACKUP FOLDER AS FileName.BK FILE
            FSys.CopyFile FileName, AppPath & NewFileName, True
        End If
    Else    'WE ARE OPENING NEW DOCUMENT MODEL
        'Opendimension = True
        NewFileName = "Model" & lDocumentCount + 1 & ".bk"
        ModelCounter = ModelCounter + 1
        CreateDB NewFileName
    End If 'Trim(FileNmme="")

    If lDocumentCount > UBound(fMainForm) Then  'WEATHER TO EXTAND ARRAY OR NOT
        ReDim Preserve fMainForm(lDocumentCount)        'YES IT IS EXTANDED TO ONE
ELEMENT
```

```
            'ACCORDING TO THE STRUCTURE OF lDcumentCout i.e. LoadUnload class
object
      Set fMainForm(lDocumentCount) = New FrmMain 'SETING NEW FORM
      Load fMainForm(lDocumentCount)
      fMainForm(lDocumentCount).MyID = lDocumentCount 'ASSIGNING FORM AN ID
      'THIS ID IS UNIQUE FOR ALL OPEN FORMS. IT IS THE INDEX OF THE IT'S REFRENCE
IN fFrmMain()
      fMainForm(lDocumentCount).Caption = "Model " & ModelCounter

   Else
      Load fMainForm(lDocumentCount)
      fMainForm(lDocumentCount).Caption = "Model " & ModelCounter
   End If
   If FileName = "" Then
      If                                    GetDimension(fMainForm(lDocumentCount),
fMainForm(lDocumentCount).PicDrawing, fMainForm(lDocumentCount).DB) = False Then
         ModelCounter = ModelCounter - 1
         ldMainForm.DeleteAtIndex CInt(lDocumentCount + 1)
         Unload fMainForm(lDocumentCount)
         Exit Sub
      End If
   End If


   If Trim(FileName) = "" Then 'NEW DOCUMENT
      fMainForm(lDocumentCount).MyID = lDocumentCount
   Else
      'To open record from database
      fMainForm(lDocumentCount).Caption = Left(NewFileName, Len(NewFileName) - 3)
      fMainForm(lDocumentCount).MyFileName = FileName
      Dim DB As Database, Rs As Recordset, I As Integer
      On Error Resume Next

      Set DB = OpenDatabase(FileName, 2, 0)
      Set Rs = DB.OpenRecordset("generators") 'OPENING AND DRAWING GENERATOR
      For I = 0 To Rs.RecordCount - 1 'INITIALIZING LOOP FOR ALL GENERATORS
         'DRAWING OBJECT ON THE NEW FORM
         r = 0
         r = Rs.Fields("Rotation")
         fMainForm(lDocumentCount).OpenToolOnForm          Rs.Fields("GeneratorsID"),
Rs.Fields("Disp_X"), Rs.Fields("Disp_Y"), Val(r), Rs.Fields("laldisp_x"), Rs.Fields("laldisp_y")
         UpdateMaintinance Rs, Rs.Fields("GeneratorsID")
         Rs.MoveNext
      Next 'END GENERATORS DRAWING
      'REST OF THE CONTROLS ARE SIMILAR
      Set Rs = DB.OpenRecordset("Feeder Parameters")
      For I = 0 To Rs.RecordCount - 1
         r = Rs.Fields("Rotation")
         fMainForm(lDocumentCount).OpenToolOnForm              Rs.Fields("FeederID"),
Rs.Fields("Disp_X"), Rs.Fields("Disp_Y"), Int(r), Rs.Fields("laldisp_x"), Rs.Fields("laldisp_y")
         UpdateMaintinance Rs, Rs.Fields("FeederID")
         Rs.MoveNext
      Next
      Set Rs = DB.OpenRecordset("Circuit Breakers")
      For I = 0 To Rs.RecordCount - 1
         fMainForm(lDocumentCount).OpenToolOnForm      Rs.Fields("CircuitbreakersID"),
Rs.Fields("Disp_X"),   Rs.Fields("Disp_Y"),   Rs.Fields("Rotation"),   Rs.Fields("laldisp_x"),
Rs.Fields("laldisp_y")
         UpdateMaintinance Rs, Rs.Fields("CircuitbreakersID")
```

```
            Rs.MoveNext
        Next
        Set Rs = DB.OpenRecordset("Transformers")
        For I = 0 To Rs.RecordCount - 1
            fMainForm(lDocumentCount).OpenToolOnForm Rs.Fields("TransformersID"),
Rs.Fields("Disp_X"), Rs.Fields("Disp_Y"), Rs.Fields("Rotation"), Rs.Fields("laldisp_x"),
Rs.Fields("laldisp_y")
            UpdateMaintinance Rs, Rs.Fields("TransformersID")
            Rs.MoveNext
        Next
        Set Rs = DB.OpenRecordset("Busbars")
        For I = 0 To Rs.RecordCount - 1
            fMainForm(lDocumentCount).OpenToolOnForm Rs.Fields("BusbarID"),
Rs.Fields("Disp_X1"), Rs.Fields("Disp_Y1"), 0, Rs.Fields("laldisp_x"), Rs.Fields("laldisp_y"),
Rs.Fields("Disp_X2"), Rs.Fields("Disp_Y2")
            UpdateMaintinance Rs, Rs.Fields("BusbarID")
            Rs.MoveNext
        Next
        Set Rs = DB.OpenRecordset("Transmission Lines")
        For I = 0 To Rs.RecordCount - 1
            fMainForm(lDocumentCount).OpenToolOnForm Rs.Fields("TransmissionLinesID"),
Rs.Fields("Disp_X1"), Rs.Fields("Disp_Y1"), 0, Rs.Fields("laldisp_x"), Rs.Fields("laldisp_y"),
Rs.Fields("Disp_X2"), Rs.Fields("Disp_Y2"), Rs.Fields("Disp_mIDX1"),
Rs.Fields("Disp_MIDY1"), Rs.Fields("Disp_mIDX2"), Rs.Fields("Disp_MIDY2")
            UpdateMaintinance Rs, Rs.Fields("TransmissionLinesID")
            Rs.MoveNext
        Next
        'fMainForm(lDocumentCount).ResizeMe
        Set Rs = DB.OpenRecordset("ModelCharacterstics", 2, 0)
        Rs.FindFirst "Attribute='Width'"
        fMainForm(lDocumentCount).PicDrawing.Width = Rs.Fields("value")
        Rs.FindFirst "attribute='height'"
        fMainForm(lDocumentCount).PicDrawing.Height = Rs.Fields("value")
        fMainForm(lDocumentCount).ResizeMe
        Rs.Close    'CLOSING TEMPRARY RECORDSET
        DB.Close    'CLOSING TEMPRARY DATABASE
        FSys.CopyFile FileName, AppPath & NewFileName, True
    End If
        fMainForm(lDocumentCount).Visible = True
        fMainForm(lDocumentCount).Show
        fMainForm(lDocumentCount).SetFocus
        ActiveFormID = lDocumentCount
End Sub
Private Function UpdateMaintinance(ByRef Rs As Recordset, ID As String)
    Dim TodayDate As New LocalDate, MDate As New LocalDate
    MDate.Value = Rs.Fields("NextMaintenanceDate")
    'Debug.Print MDate.Value
    TodayDate.Value = Day(Date) & "/" & Month(Date) & "/" & Year(Date)
    If TodayDate.CompareLocalDate(MDate) = 0 Then
        fMainForm(lDocumentCount).AddToBlinkArray ID
    ElseIf TodayDate.CompareLocalDate(MDate) = 1 Then
        If Rs.Fields("MaintenanceInterval") <> 0 Then
            Do While TodayDate.CompareLocalDate(MDate) = 1
                MDate.Value = MDate.AddDays(Rs.Fields("MaintenanceInterval"))
            Loop
            If TodayDate.CompareLocalDate(MDate) = 0 Then
fMainForm(lDocumentCount).AddToBlinkArray ID
        End If
        Rs.Edit
```

```
        Rs.Fields("NextMaintenanceDate") = MDate.Value
        Rs.Update
    Else

    End If

End Function

'THE FOLLOWING SUB SAVE ALL THE DATA OF A FORM TO IT'S DATABASE
'IT IS DEPENDENT
'   1.  rsForAll MUST SET TO THE CORRECT DATABASE AND CORRECT TABLE
'   2.  ALL THE DATA OF THE TEXT BOXES WILL BE SAVED IF
'       1.  NAME OF THE TEXT BOX START WITH "txt"
'       2.  REST OF THE NAME CONSIST OF THE FIELDNAME TO WHICH DATA IS TO STORE
Public Sub SaveFormData(Frm As Form, SAVING As RsUpdate)

    If SAVING = UpdateRecord Then   'UPDATE
        rsForAll.Edit
    ElseIf SAVING = AddNewRecord Then   'NEW ENTRY
        rsForAll.AddNew
    End If
    On Error Resume Next
    Dim ctrl As Control 'CONTROL OBJECT
    For Each ctrl In Frm    'INITALIZING LOOP FOR ALL OBJECTS ON THE FORM
        If InStr(ctrl.Name, "txt") = 1 Then 'SEARCHING FOR CONTROL WITH NAME STARTS
WITH "txt"
            'IF IT IS NOT NULL THEN STORING IT'S VALUE TO DATABASE TABLE
            If ctrl.Text <> "" Then rsForAll.Fields(Right(ctrl.Name, Len(ctrl.Name) - 3)) =
ctrl.Text
            'Debug.Print ctrl.Text
        End If
    Next
    rsForAll.Update 'UPDATION OF RECORD

End Sub

'THE FOLLOWING SUB IS LIKE THE "SaveFormData"
'IT IS ALSO DEPENDENT AS "SaveFormData"
'IT ONLY PICKS THE DATA FROM THE TABLE AND SEND IT TO THE TEXT BOXES OF THE
FORM Frm
'ACCORDING TO THE DEPENDENCY
Public Sub LoadFormData(Frm As Form)

    On Error Resume Next
    Dim ctrl As Control, Temp As New LocalDate, TempStr As String
    For Each ctrl In Frm 'INITALIZING LOOP FOR ALL OBJECTS ON THE FORM
        If InStr(ctrl.Name, "txt") = 1 Then 'SEARCHING FOR CONTROL WITH NAME STARTS
WITH "txt"
            If InStr(ctrl.Name, "Date") > 0 Then
                TempStr = ""
                'READING VALUE FROM THE RECORD AND SENDING IT TO TEXT BOX
                TempStr = Trim(rsForAll.Fields(Right(ctrl.Name, Len(ctrl.Name) - 3)))
                If TempStr = "" Then
                    ctrl.Text = "__/__/____"  'MAKING TEXT BOX EMPTY
                Else
                    Temp.Value = rsForAll.Fields(Right(ctrl.Name, Len(ctrl.Name) - 3))
                    ctrl.Text = Temp.Value
                End If
            Else
```

```
            ctrl.Text = ""  'MAKING TEXT BOX EMPTY
            'READING VALUE FROM THE RECORD AND SENDING IT TO TEXT BOX
            ctrl.Text = rsForAll.Fields(Right(ctrl.Name, Len(ctrl.Name) - 3))
          End If
          'Debug.Print Right(ctrl.Name, Len(ctrl.Name) - 3) & " = " & ctrl.Text
        End If
    Next

End Sub


Public Sub DisableAllButton()
    'This procedure will display all the tools items as unselected
    For I = 1 To 6
        MainWindow.Toolbar2.Buttons(I).MixedState = False
    Next

End Sub

Public Sub CreateDB(Name As String, Optional BackupFileName As String = "")
    'IF BACKUP FILE NAME NOT PROVIDED, THIS FUNCTION JUST COPY THE DDRCMP.sys
AS NEW DATABASE FILE WITH NAME Name(PARAMETER)
    'OTHER WISE IT COPY THE BACKUP FILE TO THE DESTINATION
    Dim FSys As New FileSystemObject
    If BackupFileName = "" Then
        'infact it copy file "DDRCMP.sys"
        FSys.CopyFile AppPath & "DDRCMP.sys", AppPath & Name
    Else
        FSys.CopyFile BackupFileName, Name, True
    End If

End Sub

Public Function GetDate(Frm As Form) As String
'THIS FUNCTION GET THE FORM AS OBJECT AND DISPLAY A CALENDER FORM ON IT'S
CENTER
'IT RETURNS THE DATE IN THE FORMAT "DD\MM\YY" AS STRING
    FrmGetDate.MonthView1 = Date
    FrmGetDate.Show 1, Frm
    If FrmGetDate.IsDateSelected Then
        GetDate      =      Format(FrmGetDate.MonthView1.Day,      "00")      &      "/"      &
Format(FrmGetDate.MonthView1.Month, "00") & "/" & FrmGetDate.MonthView1.Year
    Else
        GetDate = ""
    End If

End Function

Public Sub OpenMyInfoForm(ToolID As String, DB As Database, FrmDisInfo As FormShow)
'THIS FUNCTION IS THE MOST IMPORTANT BECAUSE IT PEFORM THE FOLLOWING TASKS
    '1. IT GET THE DATABASE OBJECT OF A MODEL, HENCE IT KNOW THE DATABASE
NAME & LOCATION
    '2. IT ANALYZ THE TOOLID AND
        'A. OPEN THE CORRESPONGING TABLE
        'B. SEARCH FOR THE TOOLID IN THE RESPECTED ID FIELD e.g. ID OF GENERATORS
ID "GeneratorsId"
        'C. IT READS THE FrmDisInfo VALUE AND
            'I.    IF IT IS ShowCharacteristics THEN IT OPEN THE BASIC PARAMETERS FORM
```

```
        'II.    IF IT ShowNone THEN IT DO NOT OPEN ANY FORM BUT JUST UPDATE THE
CONTROLS DISPLAY POSITION IN THE DATABASE
        'III.   IF IT ShowFailuresEdit THEN IT OPEN THE FAILURE FORM FOR UPDAT ONLY
        'IV.     IF IT ShowFailuresAddNew THEN IT OPEN THE FAILURE AFTER ADDING
NEW ENTRY
   If ToolID = "" Then Exit Sub    'INVALID TOOL ID
   Dim response    'FOR MESSAGE BOX
   Dim Ch1 As String * 1, Ch2 As String * 1    'TO IDINTIFY TOOL
   Ch1 = ToolID
   Ch2 = Right(ToolID, Len(ToolID) - 1)
   If Ch1 = "G" Then   'Generator

      Index = Right(ToolID, Len(ToolID) - 1) - 1  'GETING GENERATORS INDEX
      'NOW CONTROL IS IDENTIFIED

      If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
         Set rsForAll = DB.OpenRecordset("Generators", dbOpenDynaset, False) 'OPEING
ENERATORS TABLE
         rsForAll.FindFirst "GeneratorsID='" & ToolID & "'"  'SEARCHING GENERATOR ID
         If rsForAll.NoMatch Then    'NOT FOUND
            rsForAll.AddNew 'MAKING NEW ENTRY
         Else
            rsForAll.Edit   'EDIT EXISTING RECORD
         End If

         'UPDATING ID, AND NEW/OLD LOCATION ON THE FORM
         rsForAll.Fields("GeneratorsID") = ToolID
         rsForAll.Fields("DISP_X") = fMainForm(ActiveFormID).imgGenerator(Index).Left
         rsForAll.Fields("DISP_Y") = fMainForm(ActiveFormID).imgGenerator(Index).Top
         rsForAll.Fields("LALDISP_X") = fMainForm(ActiveFormID).LALGenerator(Index).Left
         rsForAll.Fields("LALDISP_Y") = fMainForm(ActiveFormID).LALGenerator(Index).Top
         rsForAll.Update

         If FrmDisInfo = ShowCharacteristics Then 'NOW WE SHALL SHOW THE
CHRACTERISTIC FORM
            rsForAll.FindFirst "GeneratorsID='" & ToolID & "'"  'SEARCHING AGAIN
            LoadFormData frmGenerator   'LOADING DATA TO FORM
            frmGenerator.Show 1 'DISPLAYING FORM AND WATING FOR CLOSE
         End If

      ElseIf FrmDisInfo = ShowFailuresAddNew Then
         Set rsForAll = DB.OpenRecordset("Generators Failures", dbOpenDynaset, False)
         rsForAll.AddNew 'MAKING NEW ENTRY TO FAILURE RECORD
         rsForAll.Fields("GeneratorsID") = ToolID    'ADDING GENERATORS ID
         rsForAll.Update 'UPDATING RECORD
         rsForAll.FindLast "GeneratorsID='" & ToolID & "'"   'SEARCHING FROM LAST
         LoadFormData FrmGeneratorFailure    'LOADING DATA IN THE FORM
         FrmGeneratorFailure.optOne = True    'MAKING OPTION ONE IS TRUE --> THE
FAILURE RECORDS OF THE GENERATOR WITH ID TOOLID ARE VIEWABLE BE DEFAULT
         FrmGeneratorFailure.Show 1 'DISPLAYING FORM AND WATING FOR CLOSE
      ElseIf FrmDisInfo = ShowFailuresEdit Then
         Set rsForAll = DB.OpenRecordset("Generators Failures", dbOpenDynaset, False)
         rsForAll.FindFirst "GeneratorsID='" & ToolID & "'" 'SEARCHING RECORD
         If rsForAll.NoMatch Then    'RECORD NOT FOUND
            'USER HAVE NO RECORD IN DATABASE AND WANTS TO EDIT A RACORD SO
WE PROMPT IT TO USER
            response = MsgBox("Failure record for " & """" & ToolID & """" & " does not
exist!" & vbNewLine & "Do you want to enter new failure record?", vbYesNo + vbQuestion,
"No Failure Record")
```

```
            If response = vbYes Then     'USER ACCEPT TO ADDNEW RECORD NOW WE
MAKE NEW ENTRY
               rsForAll.AddNew
               rsForAll.Fields("GeneratorsID") = ToolID
               rsForAll.Update
               FrmGeneratorFailure.txtGeneratorsID = ToolID
            Else
               rsForAll.Close 'OTHERWISE CLOSING RECORDSET AND
               Exit Sub 'TERMINATE SUB
            End If
         End If
         rsForAll.FindFirst "GeneratorsID='" & ToolID & "'"
         LoadFormData FrmGeneratorFailure
         FrmGeneratorFailure.optOne = True
         FrmGeneratorFailure.Show 1
      End If
   'ALL THE CONTROLS BELOW ARE SIMILAR
   ElseIf Ch1 = "C" Then   'Cricuit Breaker

      Index = Right(ToolID, Len(ToolID) - 2) - 1
      If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
         Set rsForAll = DB.OpenRecordset("Circuit Breakers", dbOpenDynaset, False)
         rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
         If rsForAll.NoMatch Then
            rsForAll.AddNew
         Else
            rsForAll.Edit
         End If
         rsForAll.Fields("CircuitbreakersID") = ToolID
         rsForAll.Fields("DISP_X") = fMainForm(ActiveFormID).imgCircuitbreaker(Index).Left
         rsForAll.Fields("DISP_Y") = fMainForm(ActiveFormID).imgCircuitbreaker(Index).Top
         rsForAll.Fields("LALDISP_X")                                          =
fMainForm(ActiveFormID).LALCircuitbreaker(Index).Left
         rsForAll.Fields("LALDISP_Y")                                          =
fMainForm(ActiveFormID).LALCircuitbreaker(Index).Top
         rsForAll.Update
         If FrmDisInfo = ShowCharacteristics Then
            rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
            LoadFormData frmCircuitbreakers
            frmCircuitbreakers.Show 1
         End If
      ElseIf FrmDisInfo = ShowFailuresAddNew Then
         Set rsForAll = DB.OpenRecordset("Circuit breakers Failures", dbOpenDynaset,
False)
         rsForAll.AddNew
         rsForAll.Fields("CircuitbreakersID") = ToolID
         rsForAll.Update
         rsForAll.FindLast "CircuitbreakersID='" & ToolID & "'"
         LoadFormData FrmCircuitbreakerFailure
         FrmCircuitbreakerFailure.optOne = True
         FrmCircuitbreakerFailure.Show 1
      ElseIf FrmDisInfo = ShowFailuresEdit Then
         Set rsForAll = DB.OpenRecordset("Circuit Breakers Failures", dbOpenDynaset,
False)
         rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
         If rsForAll.NoMatch Then
            response = MsgBox("Failure record for " & """" & ToolID & """" & " does not
exist!" & vbNewLine & "Do you want to enter new failure record?", vbYesNo + vbQuestion,
"No Failure Record")
```

```
            If response = vbYes Then
               rsForAll.AddNew
               rsForAll.Fields("CircuitbreakersID") = ToolID
               rsForAll.Update
               FrmCircuitbreakerFailure.txtCircuitBreakersID = ToolID
            Else
               rsForAll.Close
               Exit Sub
            End If
         End If
         rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
         LoadFormData FrmCircuitbreakerFailure
         FrmCircuitbreakerFailure.optOne = True
         FrmCircuitbreakerFailure.Show 1
      End If


   ElseIf Ch1 = "F" Then   'Feeder point

      Index = Right(ToolID, Len(ToolID) - 1) - 1
      If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
         Set rsForAll = DB.OpenRecordset("Feeder Parameters", dbOpenDynaset, False)
         rsForAll.FindFirst "FeederID='" & ToolID & "'"
         If rsForAll.NoMatch Then
            rsForAll.AddNew
         Else
            rsForAll.Edit
         End If
         rsForAll.Fields("DISP_X") = fMainForm(ActiveFormID).imgFeeder(Index).Left
         rsForAll.Fields("DISP_y") = fMainForm(ActiveFormID).imgFeeder(Index).Top
         rsForAll.Fields("LALDISP_X") = fMainForm(ActiveFormID).LALFeeder(Index).Left
         rsForAll.Fields("LALDISP_Y") = fMainForm(ActiveFormID).LALFeeder(Index).Top
         rsForAll.Fields("FeederID") = ToolID
         rsForAll.Update
         If FrmDisInfo = ShowCharacteristics Then
            rsForAll.FindFirst "FeederID='" & ToolID & "'"
            LoadFormData frmFeeder
            frmFeeder.Show 1
         End If
      ElseIf FrmDisInfo = ShowFailuresAddNew Or FrmDisInfo = ShowFailuresEdit Then
         MsgBox "Feeders Failure records are not available", vbExclamation, "Not Avaliable"
         Exit Sub
      End If
   ElseIf Ch1 = "B" Then   'Line Busbar

      Index = Right(ToolID, Len(ToolID) - 1) - 1
      If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
         Set rsForAll = DB.OpenRecordset("Busbars", dbOpenDynaset, False)
         rsForAll.FindFirst "BusbarID='" & ToolID & "'"
         If rsForAll.NoMatch Then
            rsForAll.AddNew
         Else
            rsForAll.Edit
         End If
         rsForAll.Fields("BusbarID") = ToolID
         rsForAll.Fields("DISP_X1") = fMainForm(ActiveFormID).LineBusbar(Index).X1
         rsForAll.Fields("DISP_Y1") = fMainForm(ActiveFormID).LineBusbar(Index).Y1
         rsForAll.Fields("DISP_X2") = fMainForm(ActiveFormID).LineBusbar(Index).X2
         rsForAll.Fields("DISP_Y2") = fMainForm(ActiveFormID).LineBusbar(Index).Y2
```

```
            rsForAll.Fields("LALDISP_X")                                    =
fMainForm(ActiveFormID).LALLineBusbar(Index).Left
            rsForAll.Fields("LALDISP_Y")                                    =
fMainForm(ActiveFormID).LALLineBusbar(Index).Top
            rsForAll.Update
            If FrmDisInfo = ShowCharacteristics Then
               rsForAll.FindFirst "BusbarID='" & ToolID & "'"
               LoadFormData frmLineBusbar
               frmLineBusbar.Show 1
            End If
         ElseIf FrmDisInfo = ShowFailuresAddNew Then
            Set rsForAll = DB.OpenRecordset("Busbars Failures", dbOpenDynaset, False)
            rsForAll.AddNew
            rsForAll.Fields("BusbarID") = ToolID
            rsForAll.Update
            rsForAll.FindLast "BusbarID='" & ToolID & "'"
            LoadFormData FrmLineBusbarFailure
            FrmLineBusbarFailure.optOne = True
            FrmLineBusbarFailure.Show 1
         ElseIf FrmDisInfo = ShowFailuresEdit Then
            Set rsForAll = DB.OpenRecordset("Busbars Failures", dbOpenDynaset, False)
            rsForAll.FindFirst "BusbarID='" & ToolID & "'"
            If rsForAll.NoMatch Then
               response = MsgBox("Failure record for " & """" & ToolID & """" & " does not
exist!" & vbNewLine & "Do you want to enter new failure record?", vbYesNo + vbQuestion,
"No Failure Record")
               If response = vbYes Then
                  rsForAll.AddNew
                  rsForAll.Fields("BusbarID") = ToolID
                  rsForAll.Update
                  FrmLineBusbarFailure.txtBusbarID = ToolID
               Else
                  rsForAll.Close
                  Exit Sub
               End If
            End If
            rsForAll.FindFirst "BusbarID='" & ToolID & "'"
            LoadFormData FrmLineBusbarFailure
            FrmLineBusbarFailure.optOne = True
            FrmLineBusbarFailure.Show 1
         End If

      ElseIf Ch1 = "T" Then
         If Ch2 = "L" Then   'Transmission Line

         Index = Right(ToolID, Len(ToolID) - 2) - 1
         If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
            Set rsForAll = DB.OpenRecordset("Transmission Lines", dbOpenDynaset, False)
            rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
            If rsForAll.NoMatch Then
               rsForAll.AddNew
            Else
               rsForAll.Edit
            End If
            rsForAll.Fields("TransmissionLinesID") = ToolID
            rsForAll.Fields("DISP_X1") = fMainForm(ActiveFormID).LineTranStart(Index).X1
            rsForAll.Fields("DISP_Y1") = fMainForm(ActiveFormID).LineTranStart(Index).Y1
            rsForAll.Fields("DISP_MidX1") = fMainForm(ActiveFormID).LineTranMid(Index).X1
            rsForAll.Fields("DISP_MidY1") = fMainForm(ActiveFormID).LineTranMid(Index).Y1
```

```
            rsForAll.Fields("DISP_MidX2") = fMainForm(ActiveFormID).LineTranMid(Index).X2
            rsForAll.Fields("DISP_MidY2") = fMainForm(ActiveFormID).LineTranMid(Index).Y2
            rsForAll.Fields("DISP_X2") = fMainForm(ActiveFormID).LineTranEnd(Index).X2
            rsForAll.Fields("DISP_Y2") = fMainForm(ActiveFormID).LineTranEnd(Index).Y2
            rsForAll.Fields("LALDISP_X")                                                    =
fMainForm(ActiveFormID).LALLineTransmission(Index).Left
            rsForAll.Fields("LALDISP_Y")                                                    =
fMainForm(ActiveFormID).LALLineTransmission(Index).Top
            rsForAll.Update
            If FrmDisInfo = ShowCharacteristics Then
               rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
               LoadFormData frmLineTransmission
               frmLineTransmission.Show 1
            End If
         ElseIf FrmDisInfo = ShowFailuresAddNew Then
            Set rsForAll = DB.OpenRecordset("Transmission Lines Failures", dbOpenDynaset,
False)
            rsForAll.AddNew
            rsForAll.Fields("TransmissionLinesID") = ToolID
            rsForAll.Update
            rsForAll.FindLast "TransmissionLinesID='" & ToolID & "'"
            LoadFormData FrmLineTransmissionFailure
            'FrmLineTranStartFailure.optOne = True
            'FrmLineTranStartFailure.Show 1
         ElseIf FrmDisInfo = ShowFailuresEdit Then
            Set rsForAll = DB.OpenRecordset("Transmission Lines Failures", dbOpenDynaset,
False)
            rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
            If rsForAll.NoMatch Then
               response = MsgBox("Failure record for " & """" & ToolID & """" & " does not
exist!" & vbNewLine & "Do you want to enter new failure record?", vbYesNo + vbQuestion,
"No Failure Record")
               If response = vbYes Then
                  rsForAll.AddNew
                  rsForAll.Fields("TransmissionLinesID") = ToolID
                  rsForAll.Update
                  FrmLineTransmissionFailure.txtTransmissionLinesID = ToolID
               Else
                  rsForAll.Close
                  Exit Sub
               End If
            End If
            rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
            LoadFormData FrmLineTransmissionFailure
            FrmLineTransmissionFailure.optOne = True
            FrmLineTransmissionFailure.Show 1
         End If

         Else       'Transformer

            Index = Right(ToolID, Len(ToolID) - 1) - 1
            If FrmDisInfo = ShowCharacteristics Or FrmDisInfo = ShowNone Then
               Set rsForAll = DB.OpenRecordset("Transformers", dbOpenDynaset, False)
               rsForAll.FindFirst "TransformersID='" & ToolID & "'"
               If rsForAll.NoMatch Then
                  rsForAll.AddNew
               Else
                  rsForAll.Edit
               End If
```

```
            rsForAll.Fields("TransformersID") = ToolID
            rsForAll.Fields("DISP_X")                                    =
fMainForm(ActiveFormID).imgTransformer(Index).Left
            rsForAll.Fields("DISP_Y")                                    =
fMainForm(ActiveFormID).imgTransformer(Index).Top
            rsForAll.Fields("LALDISP_X")                                 =
fMainForm(ActiveFormID).LALTransformer(Index).Left
            rsForAll.Fields("LALDISP_Y")                                 =
fMainForm(ActiveFormID).LALTransformer(Index).Top
            rsForAll.Update
            If FrmDisInfo = ShowCharacteristics Then
               rsForAll.FindFirst "TransformersID='" & ToolID & "'"
               LoadFormData frmTransformer
               frmTransformer.Show 1
            End If
         ElseIf FrmDisInfo = ShowFailuresAddNew Then
            Set rsForAll = DB.OpenRecordset("Transformers Failures", dbOpenDynaset,
False)
            rsForAll.AddNew
            rsForAll.Fields("TransformersID") = ToolID
            rsForAll.Update
            rsForAll.FindLast "TransformersID='" & ToolID & "'"
            LoadFormData FrmTransformerFailure
            FrmTransformerFailure.optOne = True
            FrmTransformerFailure.Show 1
         ElseIf FrmDisInfo = ShowFailuresEdit Then
            Set rsForAll = DB.OpenRecordset("Transformers Failures", dbOpenDynaset,
False)
            rsForAll.FindFirst "TransformersID='" & ToolID & "'"
            If rsForAll.NoMatch Then
               response = MsgBox("Failure record for " & """" & ToolID & """" & " does not
exist!" & vbNewLine & "Do you want to enter new failure record?", vbYesNo + vbQuestion,
"No Failure Record")
               If response = vbYes Then
                  rsForAll.AddNew
                  rsForAll.Fields("TransformersID") = ToolID
                  rsForAll.Update
                  FrmTransformerFailure.txtTransformersID = ToolID
               Else
                  rsForAll.Close
                  Exit Sub
               End If
            End If
            rsForAll.FindFirst "TransformersID='" & ToolID & "'"
            LoadFormData FrmTransformerFailure
            FrmTransformerFailure.optOne = True
            FrmTransformerFailure.Show 1
         End If
      End If
   End If
   'rsForAll.Close
End Sub


Public Sub ShowMaintenanceShudule(I As Integer)

   frmMaintenanceSchedule.Maintain fMainForm(ActiveFormID).DB, I
   frmMaintenanceSchedule.Caption = "Maintenance Schedule of " &
fMainForm(ActiveFormID).Caption
```

```
    frmMaintenanceSchedule.Show

End Sub

Public Sub PrintGrid(GRD As MSFlexGrid, ByVal LeftMargin As Long, Optional PrintLines As
Boolean = True)

    Dim curY As Long, curX As Long, curX2 As Long
    For I = 0 To GRD.Rows - 1
        curY = Printer.CurrentY + 40
        curX = LeftMargin
        For J = 0 To GRD.Cols - 1
            Dim StrTemp, TempPrint
            If PrintLines Then Printer.Line (curX, curY - 20)-(curX + GRD.ColWidth(J), curY -
20)
            If PrintLines Then Printer.Line (curX, curY - 20)-(curX, curY + 20 +
GRD.RowHeight(I))
            Printer.CurrentY = curY
            Printer.CurrentX = curX
            If IsNumeric(GRD.TextMatrix(I, J)) Then
                curX2 = Printer.CurrentX - Printer.TextWidth(GRD.TextMatrix(I, J)) +
GRD.ColWidth(J) - 30
                If InStr(GRD.TextMatrix(I, J), vbNewLine) Then
                    StrTemp = GRD.TextMatrix(I, J) & vbNewLine
                    Do While InStr(StrTemp, vbNewLine)
                        TempPrint = Left(StrTemp, InStr(StrTemp, vbNewLine) - 1)
                        StrTemp = Right(StrTemp, Len(StrTemp) - InStr(StrTemp, vbNewLine) - 1)
                        Printer.CurrentX = curX2 - 80
                        Printer.Print TempPrint
                    Loop
                Else
                    Printer.CurrentX = curX2 - 80
                    Printer.Print GRD.TextMatrix(I, J)
                End If
            ElseIf GRD.TextMatrix(I, J) = "A" Or GRD.TextMatrix(I, J) = "U" Then
                curX2 = Printer.CurrentX + (GRD.ColWidth(J) -
Printer.TextWidth(GRD.TextMatrix(I, J))) / 2
                If InStr(GRD.TextMatrix(I, J), vbNewLine) Then

                    StrTemp = GRD.TextMatrix(I, J) & vbNewLine
                    Do While InStr(StrTemp, vbNewLine)
                        TempPrint = Left(StrTemp, InStr(StrTemp, vbNewLine) - 1)
                        StrTemp = Right(StrTemp, Len(StrTemp) - InStr(StrTemp, vbNewLine) - 1)
                        Printer.CurrentX = curX2
                        Printer.Print TempPrint
                    Loop
                Else
                    Printer.CurrentX = curX2
                    Printer.Print GRD.TextMatrix(I, J)
                End If
            Else
                If InStr(GRD.TextMatrix(I, J), vbNewLine) Then
                    StrTemp = GRD.TextMatrix(I, J) & vbNewLine
                    Do While InStr(StrTemp, vbNewLine)
                        TempPrint = Left(StrTemp, InStr(StrTemp, vbNewLine) - 1)
                        StrTemp = Right(StrTemp, Len(StrTemp) - InStr(StrTemp, vbNewLine) - 1)
                        Printer.CurrentX = curX + 80
                        Printer.Print TempPrint
                    Loop
```

```
            Else
               Printer.CurrentX = curX + 80
               Printer.Print GRD.TextMatrix(I, J)
            End If
         End If

         curX = curX + GRD.ColWidth(J)
         If PrintLines Then If J = GRD.Cols - 1 Then Printer.Line (curX, curY - 20)-(curX,
curY + 20 + GRD.RowHeight(I))
      Next
      If PrintLines Then If I = GRD.Rows - 1 Then Printer.Line (LeftMargin,
Printer.CurrentY)-(curX, Printer.CurrentY)
      TopMargin = 1000
      If Printer.CurrentY > Printer.Height - 2 * TopMargin - 120 Then
         If PrintLines Then Printer.Line (LeftMargin, Printer.CurrentY)-(curX,
Printer.CurrentY)
         Printer.NewPage
         Printer.CurrentY = 1000
         Printer.CurrentX = LeftMargin
      End If
   Next

End Sub

Public Function GetDimension(Frm As Form, PIC As PictureBox, DB As Database) As
Boolean

   frmDimensions.Show 1
   If frmDimensions.ApplyChange = True Then
      GetDimension = True
      PIC.Width = frmDimensions.ComboWidth * 1440
      PIC.Height = frmDimensions.ComboHeight * 1440
      Dim Rs As Recordset
      Set Rs = DB.OpenRecordset("ModelCharacterstics", dbOpenDynaset, False)
      Rs.FindFirst "attribute='width'"
      Rs.Edit
      Rs.Fields("value") = PIC.Width
      Rs.Update
      Rs.FindFirst "attribute='height'"
      Rs.Edit
      Rs.Fields("value") = PIC.Height
      Rs.Update

      Frm.ResizeMe
   End If
   'Frm.ResizeMe

End Function

Public Sub EnableAll(Enable As Boolean)


   MainWindow.Toolbar1.Buttons(3).Enabled = Enable
   MainWindow.Toolbar1.Buttons(4).Enabled = Enable
   MainWindow.Toolbar1.Buttons(5).Enabled = Enable
   MainWindow.Toolbar1.Buttons(6).Enabled = Enable
   MainWindow.Toolbar1.Buttons(7).Enabled = Enable
   MainWindow.Toolbar1.Buttons(8).Enabled = Enable
   MainWindow.Toolbar1.Buttons(9).Enabled = Enable
```

```
    MainWindow.Toolbar4.Buttons(1).Enabled = Enable
    For I = 1 To MainWindow.Toolbar2.Buttons.count
        MainWindow.Toolbar2.Buttons(I).Enabled = Enable
    Next
    For I = 1 To MainWindow.Toolbar3.Buttons.count
        MainWindow.Toolbar3.Buttons(I).Enabled = Enable
    Next
End Sub

Public Function DisplayPriod() As Boolean
    FrmPerformanceIndecCal.Show 1
    If FrmPerformanceIndecCal.OK = True Then
        'if
        DisplayPriod = True
    Else
        DisplayPriod = False
    End If
End Function
```

B.3 Class Modules

B.3.1 DecToBin

```
Dim Bits() As Boolean


Public Property Get Bit(Position As Integer) As Boolean

    If Position <= UBound(Bits) Then
        Bit = Bits(Position)
    Else
        Bit = False
    End If

End Property


Public Sub ChangeToBinary(ByVal Value As Long)
    Dim I As Integer ', Total As Integer
    'Total = Value
    'for
    ReDim Bits(1 To 1)
    I = 1
    Do While Value > 0
        ReDim Preserve Bits(1 To I)
        If Value Mod 2 = 0 Then
            Bits(I) = False
        Else
            Bits(I) = True
        End If
        Value = Int(Value / 2#)
        I = I + 1
    Loop

End Sub
```

B.3.2 LoadUnload

```
'This class is responsible for holding
'Boolean array and provide following
'1  NewItemNumber    It return the new position of the item to be loaded
                    'If there are 12 loaded items and item number
                    '4 is deleted. Then it shall return 4 for new item
                    'otherwise 13
'2  DeleteAtIndex    it shall remove the information from the specific index
                    'and make it avaliable for the next "NewItemNumber" and if
                    'there are more avaliable numbers, it shall return the smallest
'3  TotalLoaded      It shall return total no of loaded objects

'4  Total           it shall return the maxium possible value of the index
'5  OpenNewAtIndex  It shall set the position for a object to be loaded
'6  IsExist         It returns weather an object is loaded or not

'Please Open new project and include this class. Make some expriments to get the best
'understanding with this class
```

```
Private Loaded() As Boolean
Private m_RotationAngle() As Integer
'


Private Sub Class_Initialize()

   ReDim Loaded(0)
   ReDim m_RotationAngle(0)
   Loaded(0) = False

End Sub

Private Sub Class_Terminate()

   ReDim Loaded(0)

End Sub

Public Function NewItemNumber() As Integer
   NewItemNumber = UBound(Loaded)
   Dim I As Integer
   For I = LBound(Loaded) To UBound(Loaded)
      If Loaded(I) = False Then
         NewItemNumber = I
         Loaded(I) = True
         Exit Function
      End If
   Next
   ReDim Preserve Loaded(UBound(Loaded) + 1)
   ReDim Preserve m_RotationAngle(UBound(Loaded) + 1)
   NewItemNumber = UBound(Loaded)
   Loaded(UBound(Loaded)) = True
End Function

Public Sub DeleteAtIndex(iIndex As Integer)
   Index = iIndex - 1
   If Index >= LBound(Loaded) And Index <= UBound(Loaded) Then
      Loaded(Index) = False
   Else
      MsgBox "Invalid index"
   End If
End Sub


Public Function TotalLoaded() As Integer
   TotalLoaded = 0
   Dim I As Integer
   For I = LBound(Loaded) To UBound(Loaded)
      If Loaded(I) = True Then
         TotalLoaded = TotalLoaded + 1
      End If
   Next

End Function

Public Function Total() As Integer

   Total = UBound(Loaded)
```

```
End Function


Public Function IsExist(iIndex As Integer) As Boolean
   IsExist = False
   Index = iIndex - 1
   If Index >= LBound(Loaded) And Index <= UBound(Loaded) Then
      If Loaded(Index) = True Then
         IsExist = True
      End If
   End If
End Function

Public Sub OpenNewAtIndex(iIndex As Integer)

   Index = iIndex - 1
   If Index > UBound(Loaded) Then
      ReDim Preserve Loaded(Index)
      ReDim Preserve m_RotationAngle(Index)
   End If
   Loaded(Index) = True

End Sub


Public Property Let AddRotationAngle(ByVal Index As Integer, Angle As Integer)

   m_RotationAngle(Index) = m_RotationAngle(Index) + Angle
   Do While m_RotationAngle(Index) >= 360
      m_RotationAngle(Index) = m_RotationAngle(Index) - 360
   Loop

End Property

Public Property Get RotationAngle(ByVal Index As Integer) As Integer

   RotationAngle = m_RotationAngle(Index)

End Property

Public Property Let RotationAngle(ByVal Index As Integer, ByVal vNewValue As Integer)

   m_RotationAngle(Index) = vNewValue

End Property
```

B.3.3 LocalDate

```
Private MyDate As Date

'Private Sub Class_Initialize()
   'NO NEED TO CHANGE THE DEFAULT DATE
   'IT'S DEFAULT VALUE IS
   'MyDate = "12:00:00 AM"
'End Sub

Public Property Get MyMonth() As Integer
   MyMonth = Month(MyDate)
```

```
End Property

Public Property Get MyDay() As Integer
   MyDay = Day(MyDate)
End Property

Public Property Get MyYear() As Integer
   MyYear = Year(MyDate)
End Property

Public Property Get Value() As String
   Value = Format(Day(MyDate), "00") & "/" & Format(Month(MyDate), "00") & "/" &
Format(Year(MyDate), "00")
End Property

Public Property Let Value(ByVal strDate As String)

   On Error Resume Next

   Dim D As Integer, M As Integer, Y As Integer
   D = Left(Trim(strDate), InStr(strDate, "/") - 1)
   strDate = Right(Trim(strDate), Len(strDate) - InStr(strDate, "/"))
   M = Left(Trim(strDate), InStr(strDate, "/") - 1)
   Y = Right(Trim(strDate), Len(strDate) - InStr(strDate, "/"))
   MyDate = M & "/" & D & "/" & Y

   Exit Property
Prompt:
   'MsgBox "Please sepecify date in dd/mm/yyyy format"

End Property

Public Property Get AddDays(NumberOfDays As Long) As String

   Dim TempDate As Date
   TempDate = MyDate + NumberOfDays
   AddDays = Format(Day(TempDate), "00") & "/" & Format(Month(TempDate), "00") & "/"
& Format(Year(TempDate), "00")

End Property

Public Property Get CompareDate(dDate As Date) As Integer

   If dDate = MyDate Then
      CompareDate = 0
   ElseIf MyDate < dDate Then
      CompareDate = -1
   Else
      CompareDate = 1
   End If

End Property

Public Property Get CompareLocalDate(lDate As LocalDate) As Integer

   Dim Temp As Date
   Temp = lDate.MyMonth & "/" & lDate.MyDay & "/" & lDate.MyYear
   If Temp = MyDate Then
      CompareLocalDate = 0
```

```
    ElseIf MyDate < Temp Then
        CompareLocalDate = -1
    Else
        CompareLocalDate = 1
    End If

End Property
```

B.3.4 Matrix

```
Private m_Matrix() As Variant
Private m_Rows As Integer
Private m_Cols As Integer

Private Sub Class_Initialize()
    ReDim m_Matrix(0, 0)
End Sub

Private Sub Class_Terminate()
    ReDim m_Matrix(0, 0)
End Sub

Public Property Get Rows() As Integer
    Rows = m_Rows
End Property

Public Property Get TotalRows() As Integer
    TotalRows = UBound(m_Matrix)
End Property

Public Property Get TotalCols() As Integer
    TotalCols = UBound(m_Matrix, 2)
End Property

Public Property Get Cols() As Integer
    Cols = m_Cols
End Property

Public Property Let Rows(NoOfRows As Integer)
    m_Rows = NoOfRows
End Property

Public Property Let Cols(NoOfCols As Integer)
    m_Cols = NoOfCols
End Property

Public Property Get Value() As Variant
    On Error GoTo PromptError
    Value = m_Matrix(m_Rows, m_Cols)
    Exit Property
PromptError:
    MsgBox "Error: " & Err.Number & vbNewLine & Err.Description, vbCritical, "Error"
End Property

Public Property Let Value(ByVal NewValue As Variant)
On Error GoTo PromptError
    m_Matrix(m_Rows, m_Cols) = NewValue
    Exit Property
```

```
PromptError:
   MsgBox "Error: " & Err.Number & vbNewLine & Err.Description, vbCritical, "Error"
End Property

Public Property Let ChangeValue(Row As Integer, col As Integer, NewValue As Variant)
   On Error GoTo PromptError
   m_Matrix(Row, col) = NewValue
   Exit Property
PromptError:
   MsgBox "Error: " & Err.Number & vbNewLine & Err.Description, vbCritical, "Error"
End Property

Public Property Get GetValue(Row As Integer, col As Integer) As Variant
   On Error GoTo PromptError
   GetValue = m_Matrix(Row, col)
   Exit Property
PromptError:
   MsgBox "Error: " & Err.Number & vbNewLine & Err.Description, vbCritical, "Error"
End Property

Public Sub Order(Rows As Integer, Cols As Integer)
   ReDim m_Matrix(1 To Rows, 1 To Cols)
End Sub

Public Function Determinent() As Variant
   On Error GoTo PromptError
   'Determinent = Evaluate(m_Matrix)
   Determinent = Application.MDeterm(m_Matrix)
   '"MsgBox "det = " & Determinent
   Exit Function
PromptError:
   MsgBox "Error: " & Err.Number & vbNewLine & Err.Description, vbCritical, "Error"
End Function

Public Function invmatrix() As Matrix
   Set invmatrix = New Matrix
   invmatrix.Order UBound(m_Matrix), UBound(m_Matrix)
   Dim xx As Variant
   xx = Application.MInverse(m_Matrix)
   Dim I As Integer, try As Integer, J As Integer
   For I = 1 To invmatrix.TotalRows
      For J = 1 To invmatrix.TotalCols
         invmatrix.ChangeValue(I, J) = xx(I, J)
      Next
   Next
End Function


Private Function Evaluate(Determinant() As Variant) As Variant
   'Deterninant()
   'Must be declared as Deterninant(1 to N, 1 to N)
   If UBound(Determinant) > 2 Then
      Dim counter As Integer
      Evaluate = 0
      For counter = 1 To UBound(Determinant)
         If Determinant(1, counter) <> 0 Then Evaluate = Evaluate + (-1) ^ (counter + 1)
* Determinant(1, counter) * Evaluate(PartOfDeterninant(Determinant(), 1, counter))
      Next
   ElseIf UBound(Determinant) = 2 Then
```

```
      Evaluate = Determinant(1, 1) * Determinant(2, 2) - Determinant(2, 1) *
Determinant(1, 2)
   Else
      Evaluate = Abs(Determinant(1, 1))
   End If


End Function

Private Function PartOfDeterninant(Determinant() As Variant, SkipRow As Integer, SkipCol
As Integer) As Variant()
   Dim N As Integer
   N = UBound(Determinant) - 1
   Dim PartDeterninant() As Variant
   ReDim PartDeterninant(1 To N, 1 To N)
   Dim I As Integer, J As Integer
   Dim I2 As Integer, J2 As Integer
   I2 = 1
   For I = 1 To UBound(Determinant)
      If I <> SkipRow Then
         J2 = 1
         For J = 1 To UBound(Determinant, 2)
            If J <> SkipCol Then
               PartDeterninant(I2, J2) = Determinant(I, J)
               J2 = J2 + 1
            End If
         Next
         I2 = I2 + 1
      End If
   Next
   PartOfDeterninant = PartDeterninant()
End Function

Public Function AdjointMatrix() As Matrix
   Set AdjointMatrix = New Matrix
   AdjointMatrix.Order UBound(m_Matrix), UBound(m_Matrix, 2)
   If UBound(m_Matrix) = 1 Then
      AdjointMatrix.ChangeValue(1, 1) = 1 'm_Matrix(1, 1)
   ElseIf UBound(m_Matrix) = 2 Then
      AdjointMatrix.ChangeValue(1, 1) = m_Matrix(2, 2)
      AdjointMatrix.ChangeValue(2, 2) = m_Matrix(1, 1)
      AdjointMatrix.ChangeValue(1, 2) = -m_Matrix(1, 2)
      AdjointMatrix.ChangeValue(2, 1) = -m_Matrix(2, 1)
      frmWait.lalCounter = frmWait.lalCounter + 4
   Else
   Dim I As Integer, J As Integer
   For I = 1 To UBound(m_Matrix)
      For J = 1 To UBound(m_Matrix, 2)
         AdjointMatrix.ChangeValue(I, J) = ((-1) ^ (I + J)) *
Evaluate(PartOfDeterninant(m_Matrix, I, J))
         frmWait.lalCounter = frmWait.lalCounter + 1
         frmWait.lalCounter.Refresh
      Next J
   Next I
   End If
End Function


Public Function CoFactor(I As Integer, J As Integer) As Double
```

```
    If UBound(m_Matrix) = 1 Then
        CoFactor = 1
    ElseIf UBound(m_Matrix) = 2 Then
        If I = 1 And J = 1 Then CoFactor = m_Matrix(2, 2)
        If I = 2 And J = 2 Then CoFactor = m_Matrix(1, 1)
        If I = 1 And J = 2 Then CoFactor = -m_Matrix(1, 2)
        If I = 2 And J = 1 Then CoFactor = -m_Matrix(2, 1)
    Else
        CoFactor = ((-1) ^ (I + J)) * Evaluate(PartOfDeterninant(m_Matrix, I, J))
    End If

End Function
```

B.3.5 MyClipboard

```
Dim MyData() As String
Dim MyToolID As String
Dim m_Rotation As Double
'

Public Sub SaveDataFrom(DB As Database, ToolID As String)
    MyToolID = ToolID
    If ToolID = "" Then Exit Sub    'INVALID TOOL ID
    Dim response    'FOR MESSAGE BOX
    Dim Ch1 As String * 1, Ch2 As String * 1    'TO IDINTIFY TOOL
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Set rsForAll = DB.OpenRecordset("Generators", dbOpenDynaset, False) 'OPEING
ENERATORS TABLE
        rsForAll.FindFirst "GeneratorsID='" & ToolID & "'"
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Set rsForAll = DB.OpenRecordset("Circuit Breakers", dbOpenDynaset, False)
        rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
    ElseIf Ch1 = "F" Then   'Feeder point
        Set rsForAll = DB.OpenRecordset("Feeder Parameters", dbOpenDynaset, False)
        rsForAll.FindFirst "FeederID='" & ToolID & "'"
    ElseIf Ch1 = "B" Then   'Line Busbar
        Set rsForAll = DB.OpenRecordset("Busbars", dbOpenDynaset, False)
        rsForAll.FindFirst "BusbarID='" & ToolID & "'"
    ElseIf Ch1 = "T" Then
        If Ch2 = "L" Then   'Transmission Line
            Set rsForAll = DB.OpenRecordset("Transmission Lines", dbOpenDynaset, False)
            rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
        Else
            Set rsForAll = DB.OpenRecordset("Transformers", dbOpenDynaset, False)
            rsForAll.FindFirst "TransformersID='" & ToolID & "'"
        End If
    End If
    ReDim MyData(rsForAll.Fields.count)
    m_Rotation = 0
    On Error Resume Next
    For I = 0 To UBound(MyData) - 1
        MyData(I) = 0
        MyData(I) = rsForAll.Fields(I)
        'Debug.Print rsForAll.Fields(I).Name & " " & I
```

```
        If LCase(rsForAll.Fields(I).Name) = "rotation" Then m_Rotation = MyData(I)
    Next
    rsForAll.Close

End Sub

Public Function Rotation()
    Rotation = m_Rotation
End Function

Public Function SaveDataToDB(DB As Database) As Boolean

    Dim ToolID As String
    ToolID = MyToolID
    If ToolID = "" Then Exit Function    'INVALID TOOL ID
    Dim response    'FOR MESSAGE BOX
    Dim Ch1 As String * 1, Ch2 As String * 1    'TO IDINTIFY TOOL
    Ch1 = ToolID
    Ch2 = Right(ToolID, Len(ToolID) - 1)
    If Ch1 = "G" Then   'Generator
        Set rsForAll = DB.OpenRecordset("Generators", dbOpenDynaset, False) 'OPEING
ENERATORS TABLE
        rsForAll.FindFirst "GeneratorsID='" & ToolID & "'"
    ElseIf Ch1 = "C" Then   'Cricuit Breaker
        Set rsForAll = DB.OpenRecordset("Circuit Breakers", dbOpenDynaset, False)
        rsForAll.FindFirst "CircuitbreakersID='" & ToolID & "'"
    ElseIf Ch1 = "F" Then   'Feeder point
        Set rsForAll = DB.OpenRecordset("Feeder Parameters", dbOpenDynaset, False)
        rsForAll.FindFirst "FeederID='" & ToolID & "'"
    ElseIf Ch1 = "B" Then   'Line Busbar
        Set rsForAll = DB.OpenRecordset("Busbars", dbOpenDynaset, False)
        rsForAll.FindFirst "BusbarID='" & ToolID & "'"
    ElseIf Ch1 = "T" Then
        If Ch2 = "L" Then   'Transmission Line
            Set rsForAll = DB.OpenRecordset("Transmission Lines", dbOpenDynaset, False)
            rsForAll.FindFirst "TransmissionLinesID='" & ToolID & "'"
        Else
            Set rsForAll = DB.OpenRecordset("Transformers", dbOpenDynaset, False)
            rsForAll.FindFirst "TransformersID='" & ToolID & "'"
        End If
    End If

    If rsForAll.NoMatch = True Then
        rsForAll.AddNew
        On Error Resume Next
        For I = 0 To UBound(MyData)
            rsForAll.Fields(I) = MyData(I)
        Next
        rsForAll.Update
        SaveDataToDB = True
    Else
        response = MsgBox("Are you sure! you want to replace tool with ID " & ToolID,
vbInformation + vbYesNo)
        If response = vbYes Then
            rsForAll.Edit
            On Error Resume Next
            For I = 0 To UBound(MyData)
                rsForAll.Fields(I) = MyData(I)
            Next
```

```
            rsForAll.Update
            SaveDataToDB = True
        ElseIf response = vbNo Then
            SaveDataToDB = False
        End If
    End If

    rsForAll.Close

End Function

Public Property Get ToolID() As String
    ToolID = MyToolID
End Property
```

B.3.6 Simulation

```
Private Type BusBarData
    ID As String
    Power As Double
    voltage As Double
    BusCode As Integer
End Type
Private Type TransmissionLineData
    ID As String
    Power As String
    Induction As Double
    Direction As Integer
    CapLimit As Integer
    Imagine As Double
    Reactance As Double
    ConnectedFrom As Double
    connectedto As Double
End Type
Private Type FeederData
    ID As String
    Power As String
    Reactive As String
End Type
Private Type GeneratorData
    ID As String
    Power As String
    NearestBusbar As String
    ManufacturerFailureRate As String
    ManufacturerRepairRate As String
    Reactive As String
    Qmin As String
    Qmax As String
End Type
Private Type TransformerData
    ID As String
    Power As String
End Type
Private Type CircuitBreakerData
    ID As String
    Power As String
End Type
'Types Defined
```

```
Private Busbar() As BusBarData
Private TranLine() As TransmissionLineData
Private Generator() As GeneratorData
Private Feeder() As FeederData
Private Circuitbreaker() As CircuitBreakerData
Private Transformer() As TransformerData


Public Sub Initialize(DB As Database)
    'This class is created When the Transmission Graph is complete
    'This class get data from the FrmGraphTable's grid
    On Error Resume Next

'Loading Busbar Data From Daabase ------------------------------------------------------------------
---
    Set rsForAll = DB.OpenRecordset("Busbars", dbOpenDynaset, False)
    rsForAll.MoveLast
    ReDim Busbar(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        Busbar(I).ID = rsForAll.Fields("busbarID")
        Busbar(I).Power = rsForAll.Fields("RealPower")
        Busbar(I).voltage = rsForAll.Fields("NominalVoltage")
        Busbar(I).BusCode = rsForAll.Fields("BusCode")
        rsForAll.MoveNext
    Next

'Loading Generators Data From Database -------------------------------------------------------------
---------
    Set rsForAll = DB.OpenRecordset("Generators", dbOpenDynaset, False)
    rsForAll.MoveLast
    ReDim Generator(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        Generator(I).ID = rsForAll.Fields("generatorsID")
        Generator(I).Power = rsForAll.Fields("RealPower")
        Generator(I).ManufacturerFailureRate = rsForAll.Fields("ManufacturerFailureRate")
        Generator(I).ManufacturerRepairRate = rsForAll.Fields("ManufacturerRepairRate")
        Generator(I).Reactive = rsForAll.Fields("ReactivePower")
        Generator(I).Qmin = rsForAll.Fields("Qmin")
        Generator(I).Qmax = rsForAll.Fields("Qmax")
        rsForAll.MoveNext
    Next
'Loading CircuitBreaker Data From Database ---------------------------------------------------------
-------------
    Set rsForAll = DB.OpenRecordset("Circuit Breakers", dbOpenDynaset, False)
    rsForAll.MoveLast
    ReDim Circuitbreaker(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        Circuitbreaker(I).ID = rsForAll.Fields("CircuitBreakersID")
        'CircuitBreaker(I).Power = rsForAll.Fields("RealPower")
        rsForAll.MoveNext
    Next

'Loading Feeder Data From Database ------------------------------------------------------------------
----
    Set rsForAll = DB.OpenRecordset("Feeder Parameters", dbOpenDynaset, False)
```

```
    rsForAll.MoveLast
    ReDim Feeder(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        Feeder(I).ID = rsForAll.Fields("FeederID")
        Feeder(I).Power = rsForAll.Fields("RealPower")
        Feeder(I).Reactive = rsForAll.Fields("ReactivePower")
        rsForAll.MoveNext
    Next

'Loading Transmission Line Data From Database ----------------------------------------------------
----------------
    Set rsForAll = DB.OpenRecordset("Transmission Lines", dbOpenDynaset, False)
    rsForAll.MoveLast
    ReDim TranLine(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        TranLine(I).ID = rsForAll.Fields("TransmissionLinesID")
        Dim Index As Integer
        Index = Right(TranLine(I).ID, Len(TranLine(I).ID) - 2)
        TranLine(I).Power = rsForAll.Fields("RealPower")
        TranLine(I).Induction = rsForAll.Fields("Induction")
        TranLine(I).CapLimit = rsForAll.Fields("CapLimit")
        TranLine(I).Imagine = rsForAll.Fields("TotalInductance") -
rsForAll.Fields("TotalCapacitance")
        TranLine(I).Reactance = rsForAll.Fields("TotalResistance")
        TranLine(I).ConnectedFrom = rsForAll.Fields("Connected From")
        TranLine(I).connectedto = rsForAll.Fields("Connected to")
        rsForAll.MoveNext
    Next

'Loading Transformer Data From Database ----------------------------------------------------------
-----------
    Set rsForAll = DB.OpenRecordset("Transformers", dbOpenDynaset, False)
    rsForAll.MoveLast
    ReDim Transformer(rsForAll.RecordCount)
    rsForAll.MoveFirst
    For I = 1 To rsForAll.RecordCount
        Transformer(I).ID = rsForAll.Fields("TransformersID")
        Transformer(I).Power = rsForAll.Fields("RealPower")
        rsForAll.MoveNext
    Next

    'Finding Connection of a generator to it's nearest busbar
    Dim PreTool As String, CurrentTool As String
    Dim J As Integer
    For I = 1 To UBound(Generator)
        PreTool = Generator(I).ID
        CurrentTool = PreTool
        For J = 1 To frmGraphTable.grdGraph.Rows - 1
            If SearchID(CurrentTool) Then
                frmGraphTable.grdGraph.col = 3
                If frmGraphTable.grdGraph.Text <> PreTool Then
                    If InStr(frmGraphTable.grdGraph.Text, "B") = 1 Then
                        Generator(I).NearestBusbar = frmGraphTable.grdGraph.Text
                        Exit For
                    Else
                        PreTool = CurrentTool
                        CurrentTool = frmGraphTable.grdGraph.Text
```

```
                End If
            Else
                frmGraphTable.grdGraph.col = 2
                If frmGraphTable.grdGraph.Text <> PreTool Then
                    If InStr(frmGraphTable.grdGraph.Text, "B") = 1 Then
                        Generator(I).NearestBusbar = frmGraphTable.grdGraph.Text
                        Exit For
                    Else
                        PreTool = CurrentTool
                        CurrentTool = frmGraphTable.grdGraph.Text
                    End If
                End If
            End If
            If InStr(CurrentTool, "TL") = 1 Then
                SearchID CurrentTool
                frmGraphTable.grdGraph.col = 2
                If frmGraphTable.grdGraph.Text = PreTool Then TranLineDirectionSet
CurrentTool, 1
                frmGraphTable.grdGraph.col = 3
                If frmGraphTable.grdGraph.Text = PreTool Then TranLineDirectionSet
CurrentTool, -1
            End If
        Else
            Exit For
        End If
    Next
Next

End Sub

Private Sub TranLineDirectionSet(ID As String, Value As Integer)
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            TranLine(I).Direction = Value
            Exit Sub
        End If
    Next

End Sub

Private Function SearchID(ID As String) As Boolean
    Dim I As Integer
    frmGraphTable.grdGraph.col = 1
    For I = 0 To frmGraphTable.grdGraph.Rows
        frmGraphTable.grdGraph.Row = I
        If frmGraphTable.grdGraph.Text = ID Then
            SearchID = True
            Exit For
        End If
    Next

End Function

Public Function BusBar_Net_Power(IDBusbar As String) As Double
    On Error GoTo NullExist
    BusBar_Net_Power = 0
    For I = 1 To UBound(Generator)
```

```
      If Generator(I).NearestBusbar = IDBusbar Then BusBar_Net_Power =
BusBar_Net_Power + Val(Generator(I).Power)
   Next

   For I = 1 To frmGraphTable.grdGraph.Rows - 1
      frmGraphTable.grdGraph.col = 1
      frmGraphTable.grdGraph.Row = I
      If frmGraphTable.grdGraph.Text = IDBusbar Then
         frmGraphTable.grdGraph.col = 3
         If InStr(frmGraphTable.grdGraph.Text, "F") = 1 Then
            BusBar_Net_Power = BusBar_Net_Power -
Power_Feeder(frmGraphTable.grdGraph.Text)
         End If
      End If
   Next
   Exit Function
NullExist:
   BusBar_Net_Power = 0
End Function

Public Function Tran_Line_Power(IDBusbar1 As String, IDBusbar2 As String) As Double
   Dim I As Integer, Temp As String
   Tran_Line_Power = 0
   For I = 0 To frmGraphTable.grdGraph.Rows - 1
      frmGraphTable.grdGraph.Row = I
      frmGraphTable.grdGraph.col = 1
      Temp = frmGraphTable.grdGraph.Text
      If InStr(Temp, "TL") Then
         frmGraphTable.grdGraph.col = 2
         If frmGraphTable.grdGraph.Text = IDBusbar1 Then
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = IDBusbar2 Then
               Tran_Line_Power = Power_TransmissionLine(Temp)
               Exit Function
            End If
         ElseIf frmGraphTable.grdGraph.Text = IDBusbar2 Then
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = IDBusbar1 Then
               Tran_Line_Power = Power_TransmissionLine(Temp)
               Exit Function
            End If
         End If
      End If
   Next

End Function

Public Function Tran_Line_Imagine(IDBusbar1 As String, IDBusbar2 As String) As Double
   Dim I As Integer, Temp As String
   Tran_Line_Imagine = 0
   For I = 0 To frmGraphTable.grdGraph.Rows - 1
      frmGraphTable.grdGraph.Row = I
      frmGraphTable.grdGraph.col = 1
      Temp = frmGraphTable.grdGraph.Text
      If InStr(Temp, "TL") Then
         frmGraphTable.grdGraph.col = 2
         If frmGraphTable.grdGraph.Text = IDBusbar1 Then
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = IDBusbar2 Then
```

```
                Tran_Line_Imagine = Imagine_TransmissionLine(Temp)
                Exit Function
            End If
        ElseIf frmGraphTable.grdGraph.Text = IDBusbar2 Then
            frmGraphTable.grdGraph.col = 3
            If frmGraphTable.grdGraph.Text = IDBusbar1 Then
                Tran_Line_Imagine = Imagine_TransmissionLine(Temp)
                Exit Function
            End If
        End If
    End If
Next

End Function

Public Function Tran_Line_Reactance(IDBusbar1 As String, IDBusbar2 As String) As Double
    Dim I As Integer, Temp As String
    Tran_Line_Reactance = 0
    For I = 0 To frmGraphTable.grdGraph.Rows - 1
        frmGraphTable.grdGraph.Row = I
        frmGraphTable.grdGraph.col = 1
        Temp = frmGraphTable.grdGraph.Text
        If InStr(Temp, "TL") Then
            frmGraphTable.grdGraph.col = 2
            If frmGraphTable.grdGraph.Text = IDBusbar1 Then
                frmGraphTable.grdGraph.col = 3
                If frmGraphTable.grdGraph.Text = IDBusbar2 Then
                    Tran_Line_Reactance = Reactance_TransmissionLine(Temp)
                    Exit Function
                End If
            ElseIf frmGraphTable.grdGraph.Text = IDBusbar2 Then
                frmGraphTable.grdGraph.col = 3
                If frmGraphTable.grdGraph.Text = IDBusbar1 Then
                    Tran_Line_Reactance = Reactance_TransmissionLine(Temp)
                    Exit Function
                End If
            End If
        End If
    Next

End Function

Public Function Power_Feeder(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(Feeder)
        If Feeder(I).ID = ID Then
            Power_Feeder = Feeder(I).Power
            Exit Function
        End If
    Next
End Function
Public Function Reactive_Feeder(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(Feeder)
        If Feeder(I).ID = ID Then
            Reactive_Feeder = Feeder(I).Reactive
            Exit Function
        End If
    Next
```

```
End Function
Public Function Power_Generator(ID As String) As Double
   Dim I As Integer
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Power_Generator = Val(Generator(I).Power)
         Exit Function
      End If
   Next
End Function

Public Function Reactive_Generator(ID As String) As Double
   Dim I As Integer
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Reactive_Generator = Val(Generator(I).Reactive)
         Exit Function
      End If
   Next
End Function

Public Function Qmin_Generator(ID As String) As Double
   Dim I As Integer
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Qmin_Generator = Val(Generator(I).Qmin)
         Exit Function
      End If
   Next
End Function

Public Function Qmax_Generator(ID As String) As Double
   Dim I As Integer
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Qmax_Generator = Val(Generator(I).Qmax)
         Exit Function
      End If
   Next
End Function
Property Let GeneratorManufacturerFailureRate(ID As String, Value As Double)
   Dim I As Integer
   On Error Resume Next
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Generator(I).ManufacturerFailureRate = Value
         Exit Property
      End If
   Next
End Property

Property Let GeneratorManufacturerRepairRate(ID As String, Value As Double)
   Dim I As Integer
   On Error Resume Next
   For I = 1 To UBound(Generator)
      If Generator(I).ID = ID Then
         Generator(I).ManufacturerRepairRate = Value
         Exit Property
      End If
```

```
    Next
End Property


Public Function GeneratorAvalibility(ID As String) As Double
    Dim I As Integer
    On Error Resume Next
    For I = 1 To UBound(Generator)
        If Generator(I).ID = ID Then
            GeneratorAvalibility = -1
            GeneratorAvalibility = Format(Val(Generator(I).ManufacturerRepairRate) /
(Val(Generator(I).ManufacturerRepairRate) + Val(Generator(I).ManufacturerFailureRate)),
"0.000000#")
            Exit Function
        End If
    Next
End Function

Public Function GeneratorUnavalibility(ID As String) As Double
    Dim I As Integer
    On Error Resume Next
    For I = 1 To UBound(Generator)
        If Generator(I).ID = ID Then
            GeneratorUnavalibility = -1
            GeneratorUnavalibility = Format(Val(Generator(I).ManufacturerFailureRate) /
(Val(Generator(I).ManufacturerRepairRate) + Val(Generator(I).ManufacturerFailureRate)),
"0.000000#")
            Exit Function
        End If
    Next
End Function

Public Function Power_Transformer(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(Transformer)
        If Transformer(I).ID = ID Then
            Power_Transformer = Val(Transformer(I).Power)
            Exit Function
        End If
    Next
End Function

Public Function Tran_Line_Direction(ID As String) As Integer
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Tran_Line_Direction = Val(TranLine(I).Direction)
            Exit Function
        End If
    Next
End Function

Public Function ZeroPower_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            'Power_TransmissionLine = 0
            Exit Function
        End If
```

```
    Next
End Function

Public Function Power_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Power_TransmissionLine = Val(TranLine(I).Power)
            Exit Function
        End If
    Next
End Function

Public Function Imagine_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Imagine_TransmissionLine = Val(TranLine(I).Imagine)
            Exit Function
        End If
    Next
End Function

Public Function Reactance_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Reactance_TransmissionLine = Val(TranLine(I).Reactance)
            Exit Function
        End If
    Next
End Function

Public Function connectfr_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            connectfr_TransmissionLine = Val(TranLine(I).ConnectedFrom)
            Exit Function
        End If
    Next
End Function

Public Function connectto_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            connectto_TransmissionLine = Val(TranLine(I).connectedto)
            Exit Function
        End If
    Next
End Function

Public Function buscode_busbar(ID As String) As Integer
    Dim I As Integer
    For I = 1 To UBound(Busbar)
        If Busbar(I).ID = ID Then
            buscode_busbar = Busbar(I).BusCode
            Exit Function
```

```
        End If
    Next
End Function

Public Function Power_Busbar(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(Busbar)
        If Busbar(I).ID = ID Then
            Power_Busbar = Busbar(I).Power
            Exit Function
        End If
    Next
End Function

Public Function Voltage_Busbar(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(Busbar)
        If Busbar(I).ID = ID Then
            Voltage_Busbar = Busbar(I).voltage
            Exit Function
        End If
    Next
End Function

Public Function Induction_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Induction_TransmissionLine = Val(TranLine(I).Induction)
            Exit Function
        End If
    Next
End Function

Public Function Cap_TransmissionLine(ID As String) As Double
    Dim I As Integer
    For I = 1 To UBound(TranLine)
        If TranLine(I).ID = ID Then
            Cap_TransmissionLine = Val(TranLine(I).CapLimit)
            Exit Function
        End If
    Next
End Function
```